UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Siniša Ribić

Model pretvorbe BPEL v Amazon Simple Workflow Service

MAGISTRSKO DELO

ŠTUDIJSKI PROGRAM DRUGE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: prof. dr. Matjaž Branko Jurič

Ljubljana, 2015

IZJAVA O AVTORSTVU MAGISTRSKEGA DELA

Spodaj podpisani Siniša Ribić z vpisno številko **63050098**, sem avtor magistrskega dela z naslovom:

Model pretvorbe BPEL v Amazon Simple Workflow Service.

S svojim podpisom zagotavljam, da:

- sem magistrsko delo izdelal/-a samostojno pod vodstvom mentorja prof. dr. Matjaža Branka Juriča,
- so elektronska oblika magistrskega dela, naslova (slov., angl.), povzetka (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko magistrskega dela in
- soglašam z javno objavo elektronske oblike magistrskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 24.5.2015 Podpis avtorja:

Zahvaljujem se mentorju prof. dr. Matjažu Branku Juriču za strokovno vodenje in usmerjanje ter spodbudo pri izdelavi magistrskega dela. Zahvala gre tudi asistentu dr. Sebastijanu Špragerju za pomoč in koristne nasvete pri zaključku naloge.

Iskreno bi se rad zahvalil tudi svojim staršem, sestri in ne nazadnje prijateljem za moralno podporo v času študija.

Kazalo

Kazalo slik

Kazalo tabel

Seznam uporabljenih kratic

kratica	angleško	slovensko
Apache ODE	Apache Orchestration Director Engine	Strežnik za namestitev in izvajanje poslovnih procesov
AST	Abstract Syntax Tree	Drevo abstraktne sintaktične strukture
BPEL	Business Process Execution Language	Jezik za izvajanje poslovnih procesov
BPM	Business Process Management	Upravljanje poslovnih procesov
BPMN	Business Process Model and Notation	Grafični prikaz procesov v modelu poslovnega procesa
CFC	Control-Flow Complexity metric	Metrika števila kontrolnih poti z različnimi prehodi
CFG	Control Flow Graph	Graf nadzora pretoka
DPE	Dead-Path Elimination	Odstranjevanje nedosegljive poti
http	Hypertext Transfer Protocol	Protokol za prenos hiperbesedila
IaaS	Infrastructure as a Service	Infrastruktura kot storitev
JAXB	Java Architecture for XML Binding	Knjižnica za generiranje objektov na podlagi XML – definicije
JAX-WS	Java API for XML web services	Javanski vmesnik za spletne storitve z uporabo strukture XML
JSON	JavaScript Object Notation	Notacija JavaScript objektov
JVM	Java Virtual Machine	Javanski navidezni stroj
LOC	Lines Of Code	Število vrstic programske kode
MCC	McCabe Cyclomatic complexity metric	Metrika števila kontrolnih poti procesa
NOA	Number Of Activities	Metrika števila aktivnosti
NOAC	Number Of Activities and Control-flow elements	Metrika števila aktivnosti in kontrolnih tokov v procesu
OASIS	Advanced Open Standards for the Information Society	Agencija za standardizacijo
OWFN	Open Workflow Nets	Odprto omrežje delovnih tokov
PaaS	Platform as a Service	Računalniško okolje (platforma) kot storitev
RESTful	Representation State Transfer	Slog programske arhitekture za izdelavo razširljivih spletnih storitev
SaaS	Software as a Service	Programska oprema kot storitev
SDK	Software Development Kit	Paket za razvoj programske opreme
SWF	Simple Workflow Service	Storitev delovnega toka

UML	Unified Modeling Language	Poenoteni jezik modeliranja
WFN	Workflow Nets	Mreže delovnega toka
WSDL	Web Service Definition Language	Jezik za definicijo spletnih storitev
WSFL	Web Service Flow Language	Jezik za opis toka spletnih storitev
XLANG	Web Services for Business Process Design (Microsoft)	Jezik za definiranje poslovnih procesov podjetja Microsoft
XML	Extensible Markup Language	Razširljiv označevalni jezik
XPath	XML Path Language	Jezik za poizvedbe v XML
XSD	XML Schema Definition	Definicija sheme XML

Povzetek

Izvajanje poslovnih procesov predstavlja tehnološko zapleten postopek. Za uspešno izvedbo je potrebno zadovoljiti več različnih aspektov. Z uvedbo računalništva v oblaku se izvajanje procesov seli iz lokalne infrastrukture na računalniški oblak. S tem postopkom uspešno premostimo probleme, povezane z lokalno infrastrukturo in s pridom izkoristimo vse prednosti, ki nam jih koncept računalništva v oblaku ponuja. V ta namen predlagamo nov način preslikave poslovnih procesov, s katerim poskrbimo za selitev procesa iz lokalnega okolja na računalniški oblak. Predlagani model preslikave temelji na jeziku BPEL (*Business Process Execution Language*) in pretvorbi njegovih ključnih konstruktov. Preslikane aktivnosti lahko v obliki delovnih tokov izvajamo na spletni storitvi SWF (*Simple Workflow Service*) in s tem podpremo izvajanje poslovnih procesov na računalniškem oblaku. Način preslikovanja posameznih konstruktov poslovnih procesov je odvisen od kompleksnosti aktivnosti, zato so določeni konstrukti preslikani neposredno, spet drugi pa zahtevajo dodatno dekompozicijo in preslikovanje po delih. Kompleksnejše aktivnosti so preslikane v obliki sestavljenih aktivnosti ali manjših delovnih tokov. Uspešnost preslikav posameznih aktivnosti je ovrednotena na podlagi testnega nabora poslovnih procesov. Pri uspešni verifikaciji pomemben faktor predstavlja predvsem pravilna izvedba poslovnega procesa v obliki delovnega toka spletne storitve SWF, v primerjavi z izvedbo na procesnem strežniku BPEL. Poleg verifikacije izvedbe poslovnega procesa smo preverjali tudi način izvedbe glede na zaporedje in število podanih aktivnosti in pri tem dosegli povprečno zmanjšanje števila aktivnosti za 12 %.

Ključne besede: poslovni procesi, delovni tokovi, BPEL, SWF, preslikava poslovnih procesov BPEL na delovne tokove spletne storitve SWF.

Abstract

Business processes execution is a technologically complex procedure. In order to execute processes successfully, several aspects need to be considered. With the introduction of cloud computing the process execution migrates from the local infrastructure to the cloud. By using proposed procedure we can successfully overcome the problems related to the local infrastructure and gain the advantage of all benefits provided by the cloud computing concept. For this purpose, we introduce a novel concept of business processes mapping, which takes care of process relocation from local environment to the cloud. The proposed model is based on BPEL (*Business Process Execution Language*) language and transformation of its key constructs. The execution of business processes on cloud computing can be supported by mapping activities in form of workflows on the SWF (*Simple Workflow Service)*. Mapping method of individual business process constructs depends on the complexity of the activities. Some constructs are mapped directly, while others require further decomposition and part by part mapping. Furthermore, more complex activities are mapped as composite activities or smaller workflows. Mapping performance of each activity is evaluated on a test set of business processes. The correct execution of the business process in form of SWF workflow, when compared with process execution on BPEL process server, represents an important factor for achieving successful verification. In addition to performance verification, we examined the method of implementation by sequence and number of specified activities and thus achieved an average reduction of activities by 12 %.

Keywords: business processes, workflows, BPEL, SWF, business process mapping between BPEL and SWF.

1 Uvod

Poslovni procesi predstavljajo nabor medsebojno povezanih aktivnosti, katerih cilj je doseganje zastavljenih poslovnih rezultatov [10]. Z uvedbo poslovnih procesov na več področjih se širi potreba po avtomatizmu in računalniški podpori. Vpeljevanje poslovnih procesov v podjetja ne glede na panogo velikokrat predstavlja velik napor. Rezultat uspešnosti vpeljave in načina poslovanja je viden po določenem časovnem obdobju. V veliki večini primerov je poslovanje po uvedbi računalniško podprtih poslovnih procesov uspešnejše kot pred njim [10]. Postopek vpeljave poslovnih procesov je dolgotrajen in obenem zahteva veliko vlaganja v kader in predvsem v infrastrukturo. Velik del vložka predstavlja predvsem tehnološki vidik, saj je za uspešno uvedbo in izvajanje poslovnih procesov treba vpeljati določena orodja in infrastrukturo, ki so potrebni za nemoteno izvajanje poslovnih procesov in s tem tudi uspešno poslovanje samega podjetja. Velikokrat so ti vložki previsoki in velika večina podjetij ravno iz teh razlogov obupa nad idejo o prehodu na sistematizacijo in uporabo poslovnih procesov. Na tem področju je pred nekaj leti prišlo do bistvene spremembe s pojavitvijo koncepta računalništva v oblaku. Slednji predstavlja nabor računalniških virov, ki so končnemu uporabniku dosegljivi na zahtevo. Predstavlja zaključen sklop storitev, ki se lahko dinamično prilagajajo uporabniku glede na njegove potrebe. S prihodom računalništva v oblaku je prišlo do sprememb na področju učinkovitosti uporabe računalniških virov, kar je vplivalo na olajšanje ne le finančnega vidika, pač pa predvsem tehnološkega. Tako po dolgem času niso več v prvem planu denarna sredstva, ampak predvsem način, kako in s kakšno tehnologijo priti do želenih rezultatov. Za končne uporabnike je v sklopu računalništva v oblaku predvsem zanimiv pojav ponudnikov različnih storitev. Glavni začetnik in predstavnik te ideje je podjetje Amazon. Slednje ponuja širok nabor različnih storitev IaaS (*Infrastructure as a Service*), PaaS (*Platform as a Service*) in SaaS (*Software as a Service*). Ena izmed njih je tudi spletna storitev SWF (*Simple Workflow Service*) [42], ki ponuja možnost izvajanja različnih delovnih tokov in spada v skupino storitev PaaS [24]. Spletna storitev SWF ponuja uporabniku nadzorovano izvajanje, spremljanje in preverjanje rezultatov različnih vrst delovnih tokov in njihovih opravil. Gre za storitev, ki na podlagi definiranih aktivnosti, izvedenih v določenem zaporedju in predstavljenih v obliki delovnih tokov, omogoča izvajanje le-teh na infrastrukturi računalniškega oblaka.

Izvajanje poslovnih procesov je podprto z uporabo jezik BPEL (*Business Process Execution Language*) [27], ki predstavlja standard na področju izvajanja poslovnih procesov in predpisuje definicijo zapisa poslovnega procesa [31]. Z uporabo jezika BPEL in njegovih konstruktov definiramo aktivnosti v določenem vrstnem redu, ki predstavljajo poslovni proces. Pri izvajanju slednjih v praksi velikokrat naletimo na prisotnost enega izmed večjih komercialnih ponudnikov, kot so npr. IBM, Oracle in Microsoft. Večina omenjenih ponudnikov omogoča izvajanje poslovnih procesov na lastni platformi, kar pa predstavlja

velik finančni zalogaj. Da bi uspešno premostili to oviro, smo se odločili za izdelavo modela za preslikavo jezika BPEL, ki temelji na odprtokodni rešitvi. Gre za model, ki preslika jezik BPEL v delovne tokove spletne storitve SWF, ki se izvaja na oblaku in je neodvisna od platforme ter uporabniku dosegljiva v obliki storitve PaaS. Z uporabo omenjenega modela preslikave je možen prenos vseh že obstoječih poslovnih procesov na oblak. V ta namen smo izdelali model preslikave poslovnega procesa jezika BPEL v delovne tokove, ki se izvajajo v sklopu spletne storitve SWF. V principu gre za preslikavo posameznih aktivnosti jezika BPEL v konstrukte spletne storitve SWF s poudarkom na ohranjanju enake funkcionalnosti. Izvedba preslikave poslovnega procesa je dosežena s preslikavo posameznih relevantnih konstruktov jezika BPEL v konstrukte delovnih tokov spletne storitve SWF. Pri tem smo uporabili različne pristope, saj je kompleksnost preslikav odvisna od kompleksnosti posamezne aktivnosti. Za primere primitivnih aktivnosti, med katere spadajo na primer »Receive«, »Reply«, »Invoke« in »Assign«, smo izbrali neposredno preslikavo v konstrukte delovnih tokov spletne storitve SWF, ki predstavljajo samostojno enoto primerno za večkratno uporabo. Pri kompleksnih aktivnostih, kot so na primer »If«, »While« in »RepeatUntil«, smo preslikavo izvedli na več različnih načinov glede na aktivnost. Tako smo znotraj kompleksnih aktivnosti iskali vzorce primitivnih konstruktov oz. smo kompleksno aktivnost preslikali kot gnezden delovni tok in jo poskušali ponovno razbiti na manjše primitivne aktivnosti. Na podlagi testnega nabora procesov jezika BPEL smo izvedli verifikacijo preslikav posameznih aktivnosti in s tem potrdili njihovo ustreznost. Izvajanje preslikav smo podprli tudi z implementacijo aplikacije, ki je predstavljala potrditev koncepta preslikave poslovnega procesa jezika BPEL. Vsako izmed preslikav smo ustrezno verificirali, kjer smo poleg preslikav posameznih aktivnosti izvedli tudi preslikavo testnih primerov poslovnih procesov in na podlagi rezultatov ocenili uspešnost preslikave. Ustreznost izdelane aplikacije smo potrdili s primerjavo rezultatov izvajanja testnega nabora procesov jezika BPEL na razvojnem okolju in v obliki delovnih tokov spletne storitve SWF. S tem smo pokazali, da je možno tudi obstoječe poslovne procese jezika BPEL na dokaj enostaven način preseliti v izvajalno okolje računalniškega oblaka in s tem izkoristiti vse prednosti, ki jih takšno okolje ponuja. Nabori testnih poslovnih procesov jezika BPEL, ki so prikazani v obliki slik, pa tudi vsa programska koda so zapisani v angleškem jeziku. Vse prikazane komponente in vsi modeli poslovnih procesov so bili izdelani za potrebe potrditve koncepta.

Struktura te magistrske naloge je naslednja. V drugem poglavju spoznamo jezik za izvajanje poslovnih procesov BPEL in delovne tokove spletne storitve SWF. Sledi podroben pregled področja obstoječih preslikav jezika BPEL v model BPMN, ki predstavlja grafično notacijo za modeliranje poslovnih procesov in delovnih tokov. Obenem predstavimo tudi alternativne možnosti preslikave jezika BPEL, kot je preslikava v Petrijeve mreže. Pri tem si podrobneje ogledamo tudi dva primera preslikav v OWF (*Open Workflow Nets*) in WFN (*Workflow Nets*). V četrtem poglavju sledi pregled zasnove pretvorbe jezika BPEL na delovne tokove spletne

storitve SWF, kjer na podlagi načina izvedbe poslovnih procesov predstavimo pristope k preslikavi ter zgradimo model, ki je potreben za preslikavo. Sledi podrobna analiza posameznih aktivnosti jezika BPEL in preslikava posameznih konstruktov na delovne tokove spletne storitve SWF. Peto poglavje vsebuje opis implementacije aplikacije kot primer koncepta preslikave, kjer je podrobno opisan postopek obdelave obstoječega poslovnega procesa BPEL in pripadajoče spletne storitve. Sledi opis izvedbe registracije delovnega toka in sama implementacija na spletni storitvi SWF. Za primer izvedbe je prikazan eden izmed testnih poslovnih procesov, ki vsebuje nabor določenih aktivnosti. V šestem poglavju smo izvedli verifikacijo testnih primerov in primerjali razlike med obstoječim načinom ter delovnimi tokovi spletne storitve SWF. Na koncu predstavimo ugotovitve in sklepe.

2 Poslovni procesi in delovni tokovi

Poslovni procesi se raztezajo preko različnih oddelkov znotraj organizacije. Sestavljeni so iz zbirke aktivnosti, ki predstavljajo osnovne gradnike. Glavna funkcija posamezne aktivnosti je realizacija določenih nalog in ciljev med postopkom izvedbe poslovnega procesa. Definirane poslovne procese lahko modeliramo, upravljamo in avtomatiziramo s ciljem po doseganju večje učinkovitosti, boljših zmogljivosti in optimizaciji stroškov. Na podlagi realiziranih ciljev, pridobljenih med postopkom izvajanja poslovnih procesov, pridobimo pomembne rezultate, ki končnemu uporabniku predstavljajo dodano (poslovno) vrednost. Poslovni procesi predstavljajo nadgradnjo delovnih tokov, ki so definirani kot avtomatiziran postopek izvajanja nalog na podlagi vnaprej določenih pravil in aktivnosti. Pot izvedbe delovnega toka je velikokrat vnaprej definirana, zato je izvedba različnih poti znotraj istega delovnega toka težko izvedljiva. Delovni tok omogoča interakcijo med ljudmi in aplikacijskimi sistemi ter skrbi za koordinacijo samega toka. Kot koncept je prisoten že več kot 20 let, a je z leti tudi napredoval iz tehnologije, ki je vključena v posamezne aplikacije, na nivo medprogramja (*middleware*), ki služi kot pripomoček vmesne plasti pri različnih aplikacijah neodvisno od platforme.

2.1 Jezik za izvajanje poslovnih procesov BPEL

2.1.1 Pregled jezika BPEL

Poslovni procesi definirajo način izvajanja delovnega toka predstavljenega kot zaporedje nekih aktivnosti, ki skupaj tvorijo zaključeno celoto. Za izvajanje poslovnega procesa se uporablja jezik BPEL. Ta omogoča kompozicijo, orkestracijo in koordinacijo spletnih storitev, obenem pa predstavlja najbolj razširjen in specializiran jezik za avtomatizacijo poslovnih procesov. Gre za naslednika jezika XLANG, ki je bil razvit pri podjetju Microsoft, in jezika WSFL, ki ga je razvilo podjetje IBM. Prvo različico jezika pod imenom BPEL4WS je leta 2003 standardizirala agencija za standardizacijo OASIS [26]. Trenutno je aktualna različica 2.0, ki je standardizirana leta 2004 pod uradnim imenom WS-BPEL 2.0. V osnovi jezik BPEL predstavlja jezik za orkestracijo. Koncept orkestracije definira izvršljiv proces, ki vključuje izmenjavo sporočil z zunanjimi sistemi, kjer izmenjavo sporočil nadzoruje oblikovalec orkestracije. Potek orkestracije vodi tako imenovani dirigent, nameščen v osrednjem nadzornem centru. Ta skrbi za način vodenja celotnega porazdeljenega sistema, sestavljenega iz več različnih elementov. Poslovni procesi jezika BPEL so opisani na podlagi definicije jezika v obliki notacije XML. Predstavitev poslovnega procesa, zapisanega na podlagi notacije XML, je možna tudi v grafični obliki, a le-ta ni standardizirana. S pomočjo notacije XML definiramo omejen nabor konstruktov, ki predstavljajo aktivnosti poslovnega procesa. Izvajanje poslovnih procesov poteka v posebnem okolju na ločenem procesnem strežniku. Končni uporabnik do njih dostopa preko spletnih storitev, zato večkrat slišimo, da

so poslovni procesi obravnavani kot navadne spletne storitve. Dostop do spletnih storitev s strani poslovnega procesa je zagotovljen na podlagi protokola SOAP [36] in operacij spletnih storitev, definiranih v obliki datoteke WSDL (*Web Service Definition Language*). Jezik BPEL temelji na modelu storitev WSDL 1.1, zato je dostop do drugih, vse bolj priljubljenih storitev tipa RESTful (*Representation State Transfer*), ki temeljijo na protokolu http (*Hypertext Transfer Protocol*), neizvedljiv. Za izvedbo klicev na storitve RESTful potrebujemo model storitve WSDL 2.0 (*Web Service Description Language*), ki vsebuje dodatne razširitve v obliki vezav (*WSDL 2.0 http Binding*) [31].

Jezik BPEL zagotavlja hierarhičen in grafičen kontrolni režim, ki zmanjšuje razdrobljenost poslovnega procesa ter omogoča uporabo osnovnih funkcij za manipulacijo s podatki. Nudi podporo identifikacijskemu mehanizmu za instance procesov kot tudi implicitnemu kreiranju in prekinitvam instanc procesov v sklopu življenjskega cikla poslovnega procesa. Znotraj jezika BPEL je omogočena tudi uporaba jezika za poizvedbe XPath (*XML Path Language*), ki pripomore pri luščenju podatkov v oblik notacije XML. Specifikacija BPEL omogoča uporabo dveh vrst poslovnih procesov – izvajajoči (*Executable business process*) in abstraktni (*Abstract business process*) poslovni procesi. Izvajajoči poslovni procesi predstavljajo natančno specificirane procese z vsemi podrobnostmi, ki se lahko izvajajo na procesnem strežniku jezika BPEL. Za razliko od izvajajočih so abstraktni poslovni procesi le delno definirani procesi, ki se uporabljajo za vizualni prikaz strukture in niso namenjeni izvajanju. Abstraktni procesi določajo predloge procesov ali sporočila med partnerji v procesu in ne vsebujejo informacij o dejanskem izvajanju. Poslovni procesi zapisani z jezikom BPEL, so sestavljeni iz korakov, imenovanih aktivnosti. Aktivnosti razdelimo v skupine glede na njihovo kompleksnost, tako poznamo osnovne (primitivne) in sestavljene (napredne) aktivnosti. Skupino osnovnih aktivnosti sestavljajo enostavni konstrukti, ki se uporabljajo za definiranje splošnih opravil. Med splošna opravila štejemo:

- izvajanje klicev spletnih storitev,
- čakanje na proženje začetka procesa,
- pošiljanje odgovora pri sinhronem načinu komunikacije,
- upravljanje s spremenljivkami,
- javljanje napak in izjem,
- čakanje na določen dogodek in zaključitev izvajanja procesa.

Sestavljene aktivnosti vsebujejo različne kombinacije enostavnih aktivnosti, na podlagi katerih dobimo kompleksne delovne tokove, ki specificirajo korake poslovnega procesa. Sestavljene aktivnosti tvorijo konstrukti strukturiranih aktivnosti, kot so:

- zaporedje izvajanja aktivnosti,
- tokovi različnih aktivnosti, ki se lahko izvajajo vzporedno,
- pogojno izvajanje aktivnosti,

- zanke in izbira alternativnih poti.

Poleg osnovnih in sestavljenih aktivnosti vsak proces jezika BPEL potrebuje tudi dodatne konstrukte, ki so pomembni za izvajanje poslovnih procesov. Te vrste aktivnosti smo poimenovali nastavitveno-povezovalni konstrukti, saj vsebujejo funkcionalnosti, ki so potrebne za nemoteno izvajanje vsakega procesa jezika BPEL. Gre za konstrukte, ki so zadolženi za:

- povezovanje z zunanjimi partnerji,
- uvoz podatkov iz zunanjih dokumentov,
- konstrukte, ki skrbijo za definiranje spremenljivk potrebnih za izvajanje poslovnega procesa.

V sklopu jezika BPEL je poleg naštetih definirana tudi podpora za bolj kompleksne funkcionalnosti, ki so velikokrat v uporabi v realnem svetu ob izvajanju poslovnih procesov. Med bolj kompleksne funkcionalnosti spadajo:

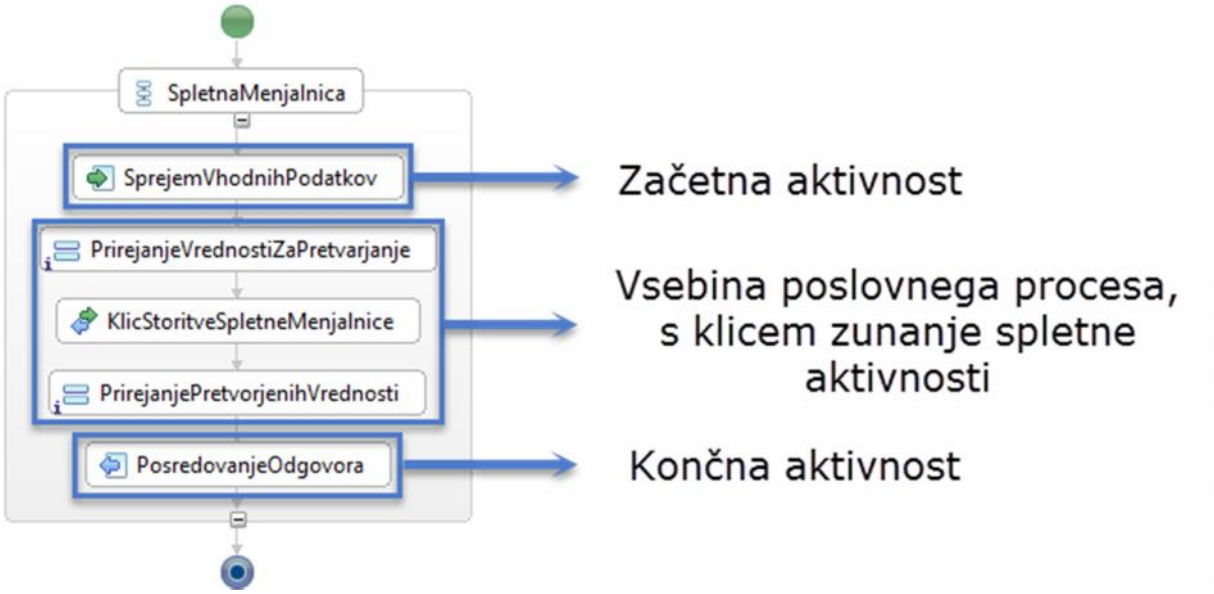
- zanke,
- zamiki,
- možnosti predčasne zaključitve procesa,
- obravnavanje napak,
- izvajanje blokov aktivnosti,
- koncept kompenzacije,
- upravljanje z dogodki,
- vzporedne aktivnosti in povezave,
- korelacija in lastnosti sporočil,
- dinamične partnerske povezave in abstraktni poslovni procesi.

Določen nabor omenjenih konstruktov, ki so med seboj povezani v neko logično zaporedje operacij, predstavlja jedro poslovnega procesa – poslovno logiko. Ob izvršitvi definiranih konstruktov se izvede določeno zaporedje aktivnosti, ki predstavlja nek poslovni proces, podprt s strani informacijskih tehnologij. Izvajanje aktivnosti poslovnega procesa poteka avtomatsko. Proženje poslovnega procesa je izvedeno v obliki kreiranja nove instance ob izvedbi določene – začetne aktivnosti. Rezultat izvedbe delovnega toka je odvisen od vhodnih podatkov in poslovne logike. Vhodni podatki običajno predstavljajo osnovo in definirajo vsebino začetne aktivnosti. Nad vhodnimi podatki se izvedejo določene operacije v sklopu poslovne logike. Število izvedenih aktivnosti in njihovo zaporedje izvajanja je odvisno od podatkov, ki se med postopkom izvajanja aktivnosti spreminjajo glede na poslovno logiko procesa. Uspešna izvedba zaporedja aktivnosti predstavlja dejanski rezultat poslovnega procesa. Posredovanje rezultata lahko izvedemo na več načinov v odvisnosti od same vsebine

poslovnega procesa. Običajno rezultat izvedbe poslovnega procesa shranimo v končno aktivnost in posredujemo končnemu uporabniku v obliki izhodnih podatkov.

2.1.2 Struktura izvajalnega procesa jezika BPEL

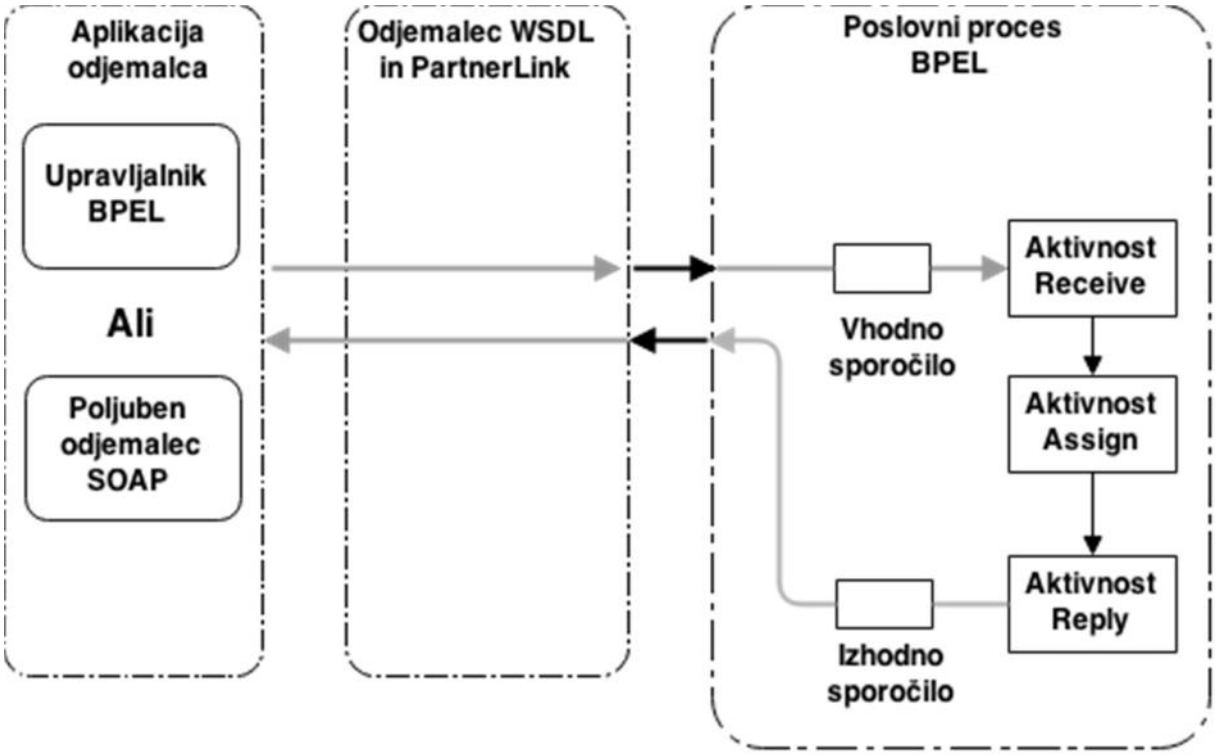
Poslovni procesi so zapisani z uporabo jezika BPEL, ki za zapis vsebuje standardizirano obliko notacije XML. Jezik BPEL predstavlja standard za definiranje akcij poslovnih procesov v povezavi s spletnimi storitvami. Poslovni proces je močno povezan z zunanjimi entitetami, s katerimi sodeluje na podlagi operacij spletnih storitev, definiranih v WSDL. Po vsem tem lahko trdimo, da tudi sam poslovni proces navzven deluje kot spletna storitev. Jezik BPEL velja za izvršljiv, vendar se poslovni procesi ne izvršijo neposredno. Začetek izvajanja poslovnega procesa je pogojen z začetkom izvajanje določene – začetne aktivnosti. Začetna aktivnost v večini primerov predstavlja neke vrste vstopno točko poslovnega procesa. Velja za vmesnik med zunanjo entiteto in poslovnim procesom. Tako pridemo do spoznanja, da za zagon posameznega procesa v večini primerov potrebujemo (zunanjo) spletno storitev, ki na podlagi proženja določene operacije (na tej storitvi) sproži začetek izvajanja poslovnega procesa. Seveda to ni vedno tako, saj so določeni poslovni procesi lahko sproženi tudi na podlagi zunanjih dogodkov, definiranih s strani drugih poslovnih procesov. Primer enostavnega poslovnega procesa je prikazan na sliki 2.1.



Slika 2.1: Primer enostavnega poslovnega procesa BPEL z začetno in končno aktivnostjo ter poslovno logiko.

Ob kreiranju poslovnih procesov se vedno kreira tudi spletna storitev v obliki datoteke WSDL, preko katere lahko sprožimo začetek izvajanja poslovnega procesa. Slika 2.2 predstavlja primer sinhronega poslovnega procesa in pripadajoče spletne storitve. Pripadajoča spletna storitev ima implementirano osnovno funkcionalnost in običajno eno samo operacijo, ki je neposredno povezana z začetno aktivnostjo poslovnega procesa. Na strani poslovnega

procesa obstaja povratna povezava s spletno storitvijo, ki je definirana z začetno aktivnostjo. Ta na nek način čaka na proženje spletne storitve, s pomočjo katere sproži zagon poslovnega procesa. Spletna storitev predstavlja vstopno točko za odjemalca, zato vsebuje tudi vhodne podatke, ki jih poda odjemalec. Vneseni podatki predstavljajo vhodne podatke, ki so potrebni za nadaljnje izvajanje poslovnega procesa in so posredovani do začetne aktivnosti. Poslovni proces ima lahko definirano eno samo vhodno točko in več možnih izhodnih točk, odvisnih od poteka poslovnega procesa. S tehničnega stališča gledano to pomeni, da se ob proženju spletne storitve v ozadju izvrši nek poslovni proces, rezultat poslovnega procesa pa je odjemalcu predstavljen v obliki odgovora pripadajoče spletne storitve. S tem dejanjem nekako skrijemo dogajanje in sam pojem poslovnih procesov, zato ima končni uporabnik občutek, da so poslovni procesi dejansko spletne storitve. Izvajanje procesa jezika BPEL poteka na dva načina – sinhron ali asinhron način, kar tudi vpliva na čas, potreben za izvedbo poslovnega procesa. Pri sinhronem načinu proženja spletne storitve gre za princip zahteva/odgovor, kjer se po pošiljanju zahteve na spletno storitev izvajanje procesa ustavi in čaka na sprejem odgovora. Storitve, ki uporabljajo tak način komunikacije, imajo običajno kratek izvajalni čas in so sposobne posredovati odgovor v prav tako kratkem času. Asinhron način komunikacije je v uporabi, ko aktivnosti opravljajo operacije, ki imajo daljši izvajalni čas. V teh primerih je čakanje na njihov zaključek časovno potratno, zato se v teh primerih odgovorov ne čaka, izvajanje procesa pa se medtem nadaljuje nemoteno. Ko se operacija na strani spletne storitve zaključi, se odgovor poslovnemu procesu pošlje v obliki proženja za to namenskih operacij.



Slika 2.2: Pripadajoča spletna storitev in poslovni proces jezika BPEL na primeru sinhrone komunikacije.

Po uspešnem kreiranju poslovnih procesov in pripadajoče spletne storitve, ki bo poskrbela za začetek poslovnega procesa, je za njegovo izvajanje potrebna še namestitev procesa v izvajalno okolje. Izvajalno okolje poslovnega procesa predstavlja spletni strežnik, kjer se izvrši celoten proces. Strežnik nudi vse potrebne storitve in vire za nemoteno izvajanje poslovne logike procesa. V primeru, da se med izvajanjem poslovnega procesa izvrši klic na zunanjo entiteto, je za operativni del zadolžena določena plast na strežniški strani, ki poskrbi za pravilno komunikacijo in izvedbo samega klica. Podobno velja za primere, ko pride do napake pri čakanju na odgovor ali druge napake med samim izvajanjem procesa. Poleg samega poslovnega procesa je na omenjen spletni strežnik nameščena tudi spremljajoča spletna storitev, ki služi za zagon izvajanja poslovnega procesa.

Testiranje poslovnih procesov lahko poteka kar preko brskalnika, s katerim dostopamo do pripadajoče spletne storitve, preko katere sprožimo začetek izvajanja poslovnega procesa. Seveda je treba pred tem zagotoviti implementacijo poslovnega procesa, njegovo vhodno točko in spremljajočo spletno storitev. Tako poslovni proces kot tudi njegovo spremljajočo spletno storitev namestimo na spletni strežnik, do katerega kasneje dostopamo. Poslovni proces sprožimo tako, da obiščemo spletno stran, kjer se nahaja pripadajoča spletna storitev, vnesemo potrebne vhodne podatke in pošljemo sporočilo. Na podlagi zahteve se v ozadju sproži izvajanje poslovnega procesa, čigar rezultat se prikaže kot odgovor prej sprožene spletne storitve. Če med izvajanjem pride do težav, namesto odgovora v obliki vrednosti dobimo kot odgovor obvestilo o napaki.

2.2 Spletna storitev za izvajanje delovnih tokov SWF

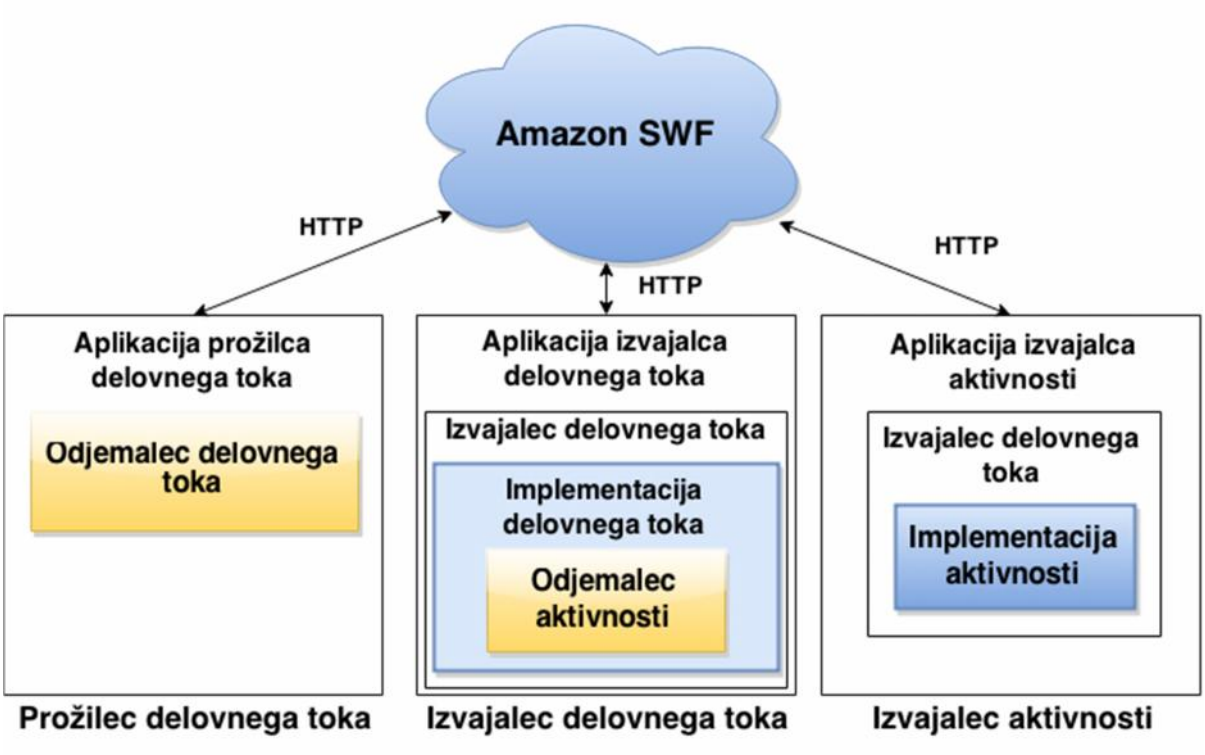
2.2.1 Pregled spletne storitve za izvajanje delovnih tokov SWF

Podjetje Amazon velja za enega od začetnikov koncepta računalništva v oblaku. V principu gre za velik nabor računalniških virov, ki so odjemalcu dosegljivi na zahtevo. V nasprotju s tradicionalnim načinom dela, kjer vsako podjetje ali ustanova za izvajanje operacij na računalniški infrastrukturi potrebuje svojo lastno računalniško infrastrukturo, računalništvo v oblaku nudi tako rekoč neomejen nabor virov. Gledano s tehnološkega stališča je pojem neomejenega nabora računalniških virov nemogoč, a odjemalec virov dobi tako izkušnjo ob njihovi uporabi. Veliko prednost računalništva v oblaku predstavlja tudi tako imenovani koncept virov na zahtevo – »on-demand«, kjer naročnik lahko poseže po računalniških virih glede na lastne potrebe in jih po potrebi poveča oz. zmanjša. Podobno velja tudi za podjetje Amazon [6], ki naročniku ponuja uporabo različnih računalniških virov, porazdeljenih po različnih regijah po vsem svetu. Trenutno je naročnikom na razpolago enajst različnih lokacij, razporejenih po celem svetu, kjer se nahajajo množice strežnikov, ki predstavljajo računalniški oblak. Kot vodilno podjetje na tem področju nudi širok nabor različnih storitev, ki so odjemalcem na voljo v različnih oblikah. Svoje storitve so razvrstili v tri glavne skupine. Skupina storitev IaaS uporabnikom omogoča najem infrastrukture in opreme v obliki storitve.

Pri tem gre konkretno za možnosti najema razne strojne opreme, strežnikov omrežja in omrežnih komponent, pri čemer je vsa oprema na strani ponudnika. PaaS predstavlja storitve, ki so uporabniku na voljo v obliki platforme. V to skupino spada najem različne strojne in omrežne opreme, virtualnih računalnikov z nameščenimi operacijskimi sistemi, ki so uporabniku na voljo v obliki platforme [24]. Skupina storitev SaaS predstavlja skupino, ki ponuja uporabo različnih programov in aplikacij v obliki storitev, ki so uporabnikom dostopne preko interneta. Vse tri zgoraj omenjene skupine delujejo na principu storitev, ki so na voljo s strani ponudnika, naročnik storitev pa deluje kot odjemalec. Ena takih je tudi storitev SWF, ki predstavlja storitev za orkestracijo in izdelavo razširljivih porazdeljenih aplikacij in pripada skupini storitev PaaS. Gre za koncept, kjer je izvajanje aplikacije odjemalcu na voljo kot spletna storitev, ki jo lahko uporablja na različnih napravah za različne potrebe. Spletna storitev SWF ponuja enostavno koordinacijo nalog in upravljanja s stanji aplikacije, ki se poleg ostalih storitev izvaja v sklopu računalniškega oblaka podjetja Amazon. Tak koncept uporabe storitev prinaša veliko prednosti, saj ponudnik storitve odjemalcu zagotavlja nemoteno delovanje storitve, obenem pa je zadolžen tudi za njeno upravljanje [24].

2.2.2 Način izvajanja delovnih tokov spletne storitve SWF

Vsaka aplikacija je sestavljena iz več različnih zadolžitev oz. nalog, ki morajo biti izvedene v določenem zaporedju na podlagi nabora dinamičnih pogojev. Spletna storitev SWF nam omogoča izvajanje delovnih tokov, kjer so posamezne aktivnosti izvedene v določenem (pravilnem) vrstnem redu. Pri izvajanju posameznih aktivnosti omenjena spletna storitev skrbi za enakomerno obremenitev posameznih konstruktov glede na medsebojne odvisnosti. Koncept spletne storitve SWF temelji na treh glavnih gradnikih, s katerimi lahko podpremo izvajanje aplikacije v obliki delovnega toka (Slika 2.3). Prvi izmed gradnikov predstavlja začetek izvajanja delovnega toka in deluje kot prožilec delovnega toka (*workflow starter*). Na podlagi proženja delovnega toka se izvede drug konstrukt – izvajalec delovnega toka (*workflow worker*), ki je zadolžen za izvajanje celotnega delovnega toka. Delovni tok sestavljajo posamezne aktivnosti, ki se izvršijo s pomočjo izvajalca aktivnosti (*activity worker*), ki predstavlja zadnjega od treh omenjenih gradnikov. Vsakega izmed treh gradnikov je smiselno izdelati v obliki ločene komponente, ki se lahko izvajajo na različnih strežnikih neodvisno od lokacije. Vsi trije gradniki komunicirajo s spletno storitvijo SWF na podlagi pošiljanja zahtevkov in prejemanja odgovorov preko protokola http. Spletna storitev SWF pri tem skrbi za izvajanje delovnega toka na podlagi seznama aktivnosti, ki preko izvajalca delovnega toka prožijo izvajanje aktivnosti. Za samo izvedbo aktivnosti je nato zadolžen izvajalec aktivnosti. Poleg izvajanja je spletna storitev SWF zadolžena tudi za hranjenje zgodovine izvedenih aktivnosti na nivoju vsakega sproženega delovnega toka.



Slika 2.3: Spletna storitev SWF in gradniki: prožilec in izvajalec delovnega toka ter izvajalec aktivnosti.

Vsak izmed gradnikov je zadolžen za izvedbo različnih opravil, zato se tudi naloge posameznih gradnikov med seboj razlikujejo. Izvajalec aktivnosti je zadolžen za izvedbo različnih delovnih nalog (predstavljenih v obliki aktivnosti – npr. »Assign«), ki morajo biti izvršene med postopkom izvajanja delovnega toka. Slednji v tem primeru predstavlja npr. poslovni proces spletne menjalnice (poglavje 3, slika 3.2) izvajalec aktivnosti pa izvrši posamezno aktivnost – prirejanje vrednosti. Vloga izvajalca aktivnosti je sestavljena iz implementacije odločitev različnih aktivnosti, ki se izvršijo in samega procesa izvedbe posamezne aktivnosti. Izvajalec aktivnosti na podlagi komunikacije s spletno storitvijo SWF pridobiva podrobnosti o posameznih razpoložljivih aktivnostih (npr. »Assign«, »If«, »While«). Če je zahtevana aktivnost razpoložljiva, prejme s strani spletne storitve SWF potrebne podatke za izvršitev želene aktivnosti. Izvajalec na podlagi prejetih podatkov izvrši aktivnost in posreduje pridobljene rezultate spletni storitvi. Tu nastopi izvajalec delovnega toka z vlogo izvršitelja orkestracije, ki skrbi za izvajanje različnih aktivnosti in podatkovnih tokov. Izvajalec delovnega toka vsebuje tudi odjemalca aktivnosti, ki je zadolžen za izvršitev nalog izvajalca aktivnosti. Tipično so te naloge izvršene v asinhronem načinu, saj na ta način zagotovljen pravilen vrstni red izvajanja aktivnosti. Komunikacija s spletno storitvijo SWF poteka na enak način kot pri izvajalcu aktivnosti, le da se v tem primeru izmenjujejo podatki o nalogah v obliki konstrukta odločevalca (*decider*), ki skrbi za vrstni red izvedbe nalog delovnega toka. Na primeru poslovnega procesa spletne menjalnice izvajalec delovnega toka poskrbi za izvajanje posameznih aktivnosti v pravilnem vrstnem redu (prirejanje podatkov,

sprožitev klica zunanje spletne storitve itd.). Za začetek izvajanja delovnega toka je zadolžena vloga prožilca delovnega toka. Ta na podlagi odjemalca sproži izvedbo nove instance delovnega toka, s katerim komunicira tudi med samim izvajanjem. Sprožilec je lahko izdelan na različne načine, v obliki namizne ali spletne aplikacije. Komunikacija med aktivnostmi in delovnim tokom poteka v asinhronem načinu, zato je dostop do podatkov implementiran v obliki objektov zagotovila za odgovor (*Promise*). Ti predstavljajo edini veljaven način izmenjave podatkov, saj za razliko od drugih zagotavljajo dostopnost do podatkov, ko so le-ti na voljo. Uporaba objektov zagotovila za odgovor je pomembna predvsem zaradi samega načina komunikacije. Tako pri asinhroni komunikaciji ne čakamo na odgovor strežnika po opravljeni zahtevi, ampak nadaljujemo z izvajanjem drugih aktivnosti. Dostop do podatkov je v tem primeru omogočen le po prejetem odgovoru s strani strežnika, za kar je zadolžen omenjen objekt, ki omogoča dostop do podatkov le ob njihovi dejanski dosegljivosti. V primeru spletne menjalnice je vloga sprožilca delovnega toka izvedena v obliki testne namizne aplikacije.

Izvedba posamezne aplikacije v obliki delovnih tokov spletne storitve SWF lahko zajema številne različne aktivnosti, ki so med seboj povezane v nek delovni tok. Za uspešno izvedbo posameznega delovnega toka in njegovega zaporedja aktivnosti je potrebna predhodna registracija tako aktivnosti kot tudi samega delovnega toka. Po uspešni izvedbi registracije aktivnosti in delovnega toka so le-te na voljo za izvajanje v okviru spletne storitve SWF. Posamezna aktivnost je v sklopu te spletne storitve predstavljena kot delovna naloga, medtem ko je delovni tok predstavljen kot zaporedje delovnih nalog. Izvajanje posameznih aktivnosti je odvisno od njene funkcionalnosti, ki se razlikuje od aktivnosti do aktivnosti. Določene aktivnosti so sestavljene iz različnih nalog, ki se lahko izvajajo na oddaljenih lokacijah, njihovo izvajanje pa lahko traja dolgo časa ali pa se celo nikoli ne izvrši (zaradi prekinitve in podobno). Več sestavljenih aktivnosti skupaj tvori poslovno logiko aplikacije, kjer je uspešnost izvedbe aplikacije pogojena s pravilnim zaporedjem izvajanja posameznih aktivnosti. Zaporedje izvajanja aktivnosti je odvisno od dinamičnih pogojev, ki so zadolženi za koordinacijo delovnih nalog na podlagi določenih pravil. Spletna storitev SWF skrbi za vsako tako nalogo, spremlja njeno dogajanje, beleži trenutno stanje in preko procesa izvajalcev aktivnosti sproži začetek izvajanja takoj, ko je to izvedljivo. Vsi podatki o izvajanju posameznih aktivnosti in delovnem toku so uporabniku na voljo preko nadzorne plošče. Slednja hrani vse podatke o posameznih registriranih aktivnostih, delovnih tokovih in njihovih različicah, kot tudi zgodovini izvajanja delovnih tokov in njihovih aktivnostih. Dostop do spletne storitve SWF (v obliki nadzorne plošče) je tako kot do večine ostalih aplikacij omogočen preko spletne aplikacije z uporabniškim računom. Na podlagi registriranega uporabniškega računa nadziramo potek in izvajanje delovnih tokov. Na ta način lahko na enem mestu spremljamo in pridobivamo vse potrebne podatke o izvajanju delovnih tokov v okviru spletne storitve SWF.

3 Pregled sorodnih raziskav s področja preslikav jezika BPEL

Upravljanje poslovnih procesov (*BPM – Business Process Management*) predstavlja zaključen življenjski cikel poslovnih procesov, sestavljen iz različnih korakov: modeliranje, kompozicija, izvajanje in spremljanje ter optimizacija [10]. Posamezen korak v življenjskem ciklu poslovnega procesa pokriva določeno področje iz določenega vidika. Razhod med posameznimi vidiki je najbolje viden pri prehodu med koraki modeliranja in kompozicije, saj se pri posameznem koraku uporabljajo različni modelirni jeziki, kar pripelje do konceptualnega razhoda. V ta namen je bila razvita notacija BPMN (*Business Process Modeling Notation*), ki naj bi poskrbela za premostitev nastalih težav s predložitvijo standardizirane vizualne notacije za procese zapisane v jeziku BPEL. Poleg same definicije notacija BPMN poskrbi tudi za definicijo preslikave med notacijo BPMN in jezikom BPEL [9]. Oba predstavljata pomembna koncepta na področju modeliranja poslovnih procesov, zato je kompatibilnost med njima v smislu preslikave konstruktov pomembna. Na to temo je bilo izdelanih veliko študij in raziskav [22, 23, 29], na podlagi katerih so pridobljeni tudi različni rezultati glede na potrebe in zahteve.

Jezik BPEL v osnovi predstavlja enostavno notacijo, a vendar njena kompleksnost narašča s kompleksnostjo samega modela poslovnega procesa. Tako slej ko prej pridemo do zapletenih poslovnih procesov in potrebe po formalizaciji. Da bi lahko ohranili kompleksnost in obenem poenostavili zapis procesa, je bilo izdelanih kar nekaj preslikav, ki so večinoma temeljile na konceptu Petrijevih mrež. Petrijeve mreže predstavljajo matematični model jezika za opis porazdeljenih sistemov. Predstavljene so v obliki usmerjenih dvostranskih grafov, kjer vozlišča grafa predstavljajo prehode (*transitions*) in prostore (*places*). Prehodi so predstavniki dogodkov, označenih kot vodoravne črte, prostori pa so označeni s krogi in predstavljajo pogoje (*conditions*). Konstrukti so med seboj povezani z usmerjenimi puščicami, ki predstavljajo loke (*arches*). Ti so lahko usmerjeni samo iz prostorov proti prehodu in obratno. Prostor, iz katerega poteka usmerjen lok proti prehodu, je poimenovan kot vhodno mesto (*input place*) prehoda. Izhodno mesto (*output place*) pa je predstavljeno kot prostor, proti kateremu je usmerjen lok iz prehoda.

Na podlagi teh osnovnih pravil je možno kreirati aktivnosti poslovnega procesa, ki so zgrajene iz enostavnih konstruktov in obenem ohranjajo funkcionalnost. Petrijeve mreže obravnavajo široko področje: BPM, načrtovanje programske opreme, umetna inteligenca itd. V ta namen obstajajo številne izpeljanke: visokonivojske (*High-level Petri nets*), časovno omejene (*Timed Petri nets*), objektno usmerjene (*Object-Oriented Petri nets*) v obliki različnih notacij, ki nudijo podporo pri preslikavi delovnih tokov.

Izvajanje poslovnih procesov z uporabo jezika BPEL predstavlja standard na tem področju. A kljub temu se je za pojavila potreba po preslikavah slednjega v druge oblike, s katerimi

prilagodimo oz. preslikamo poslovni proces in s tem podpremo drug vidik. Na tem področju prednjačita predvsem dve omenjeni preslikavi – jezik BPEL v notacijo BPMN in Petrijeve mreže, kjer je vsaka osredotoča na reševanje določenih problemov. Oba načina preslikave sta podrobneje opisana v nadaljnjih poglavjih (3.1 in 3.2). Pri zasnovi koncepta preslikave jezika BPEL v delovne tokove spletne storitve SWF nobena od omenjenih preslikav ni primerna, zato smo se odločili za izdelavo lastne preslikave.

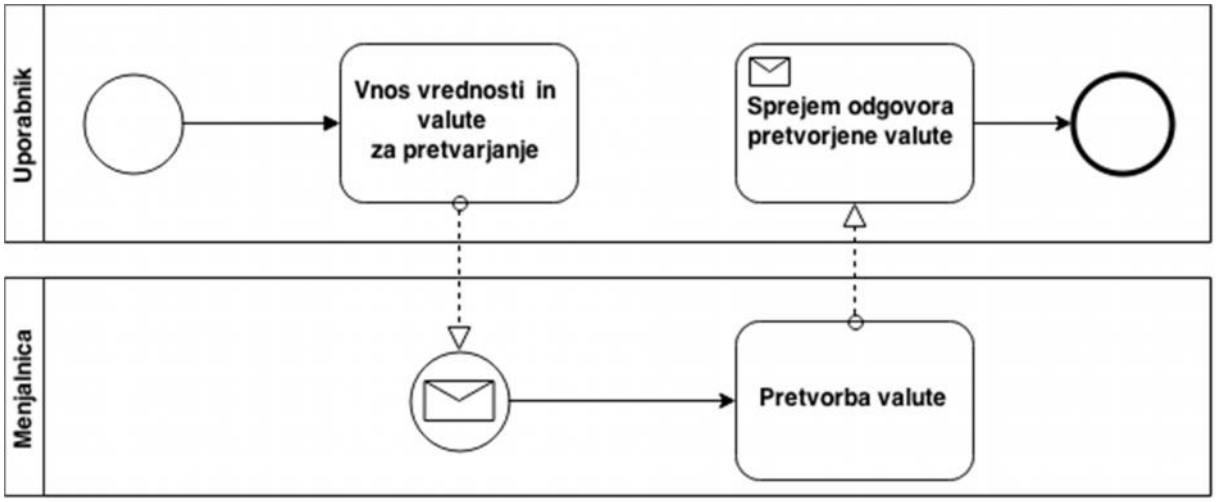
3.1 Preslikava jezika BPEL v notacijo BPMN

3.1.1 Umestitev notacije BPMN na podlagi jezika BPEL

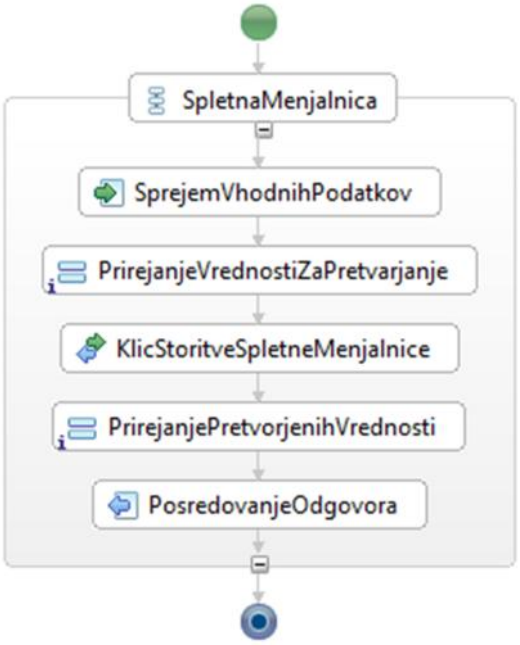
Notacija BPMN predstavlja standard za modeliranje poslovnih procesov in omogoča grafični zapis aktivnosti poslovnega procesa. Temelji na tehniki diagrama toka podatkov (podobno kot diagram aktivnosti) in je izdelan na podlagi poenotenega jezika modeliranja – UML. Glavni namen diagrama notacije BPMN je podpora upravljanju poslovnih procesov tako za poslovne kot tudi za bolj tehnično usmerjene uporabnike na podlagi notacije, ki je intuitivna in obenem sposobna prikazati bolj kompleksno semantiko procesa. Notacija BPMN je standardizirana v več različicah, kar je omogočilo njeno evolucijo iz modelirnega jezika v izvršljiv jezik. Začetna različica notacije BPMN z oznako 1.0 (*Business Process Modeling Notation*) se osredotoča predvsem na definicijo in pomenoslovje diagrama poslovnega procesa ter združitev nabora dobrih praks. Poleg definicije ključni cilj omenjene različice predstavljala tudi vizualizacija poslovno-orientirane notacije jezika BPEL4WS (zapisanega v obliki notacije XML), namenjenega izvajanju poslovnih procesov [9]. Trenutno aktualna različica notacije BPMN – 2.0 (*Business Process Model and Notation*: v nadaljevanju BPMN) – je poleg sprememb in formalne definicije v obliki metamodela poskrbela tudi za poenotenje več predhodnih različic. Ključne spremembe so vidne v obliki spremembe notacije, ki je bila dopolnjena z novimi konstrukti in diagrami (pogovor in koreografija), vpeljavo novih dogodkovnih podprocesov in podporo neprekinitvenim dogodkom, kot tudi z razširitvami mehanizmov procesnega modela in grafične notacije. Ključno prednost nove različice predstavlja normalizacija modela v obliki shem notacije XML in možnost neposredne implementacije oz. pretvorbe obstoječega modela v drugo razširljivo obliko.

Pri definiciji in načrtovanju poslovnih procesov velikokrat naletimo na dilemo, ki je odvisna od vidika opisovanja nekega poslovnega procesa. Prednjačita dva različna načina opisa poslovnih procesov, procesni model BPMN (Slika 3.1) in izvršljiv model oz. jezik BPEL (Slika 3.2). Jezik BPEL predstavlja standard za implementacijo procesa v storitveno usmerjeni arhitekturi, medtem ko je model BPMN namenjen predstavitvi oz. načrtu poslovnega procesa. Opisa se med seboj razlikujeta, saj predstavljata različne poglede in koncepte. Procesni model BPMN je namenjen opisu poslovnega procesa v obliki grafa, kjer je poslovni proces predstavljen na abstrakten način (s pomočjo grafov). Nasprotno od tega je

jezik BPEL blokovno strukturiran jezik, kjer je poslovni proces predstavljen s shemo XML, ki je vnaprej določena z definirano predlogo XSD.



Slika 3.1: Primer poslovnega procesa modela BPMN, ki prikazuje funkcionalnost spletne menjalnice.



Slika 3.2: Primer poslovnega procesa BPEL, ki prikazuje funkcionalnost spletne menjalnice.

V obeh primerih gre za modele poslovnega procesa, zato je preslikava med njima več kot dobrodošla. Konceptualno neskladje med notacijo BPMN in jezikom BPEL poveča kompleksnost preslikave, na kar nakazujeta tudi Reckler in Mendling [33]. V določenih primerih zato preslikava iz notacije BPMN v jezik BPEL ni izvedljiva, medtem ko preslikava v obratni smeri v splošnem ne predstavlja tako zahtevnega problema [17]. Notacija BPMN omogoča veliko večji nabor konstruktov kot jezik BPEL, kar s tehničnega vidika olajša postopek preslikave, a je kljub temu ta vse prej kot trivialna. Omejitve preslikav med dvema

modeloma so tudi pogosta tema razprav v akademskih krogih, ki so vodile do bolj prefinjenih načinov transformacij. Nastalo je veliko raziskav (prepoznavanje vzorcev notacije BPMN, formalna verifikacija diagramov, preslikava v Petrijeve mreže …) [17] o različnih konceptih in pristopih pri izvedbi same preslikave.

V primeru neposredne preslikave med jezikom BPEL in notacijo BPMN je potrebno definirati nekaj pravil in pri tem sprejeti nekaj kompromisov. Neposredna preslikava modela BPMN v jezik BPEL zahteva dvosmerno poravnavo tako notacije kot jezika. Posredovanje informacij o samem procesu bi v tem primeru omogočili z uporabo vizualizacije v obliki standardizirane notacije [41]. Kljub temu popolna dvosmerna povezava med notacijo BPMN in jezikom BPEL ne obstaja. Približek temu predstavlja koncept modeliranja jezika BPEL z uporabo grafičnih elementov notacije BPMN, kot je predstavljen v [35]. Izdelane obstoječe različice preslikav med jezikom BPEL in notacijo BPMN niso popolne [38], saj ne pokrivajo polnega nabora jezika BPEL.

3.1.2 Preslikava aktivnosti jezika BPEL

Preslikavo procesa štejemo za uspešno, če je vsak konstrukt jezika BPEL možno preslikati v model notacije BPMN. Konstrukte oziroma aktivnosti jezika BPEL razdelimo glede na funkcionalnost v manjše skupine, in sicer na enostavne – primitivne, napredne – kompleksne in generične aktivnosti. V skupino enostavnih aktivnosti tako spadajo:

- »Invoke«,
- »Receive«,
- »Reply«,
- »Validate«,
- »Assign«,
- »Wait,
- »Exit«,
- »Throw«,
- »Rethrow«,
- »Compensate«,
- »CompensateScope«,

kjer pa izmed naštetih neposredno lahko preslikamo le aktivnosti »Receive«, »Wait« in »Exit«. Ostale naštete aktivnosti je možno le delno preslikati zaradi omejitev, ki so pri posameznih konstruktih opisane v nadaljevanju poglavja. Za določene konstrukte kot npr. »Validate« preslikava zaradi pomanjkanja koncepta pri modelu BPMN ni izvedljiva.

Za vsako aktivnost jezika BPEL, kjer je neposredna preslikava izvedljiva, pomeni, da v modelu notacije BPMN obstaja konstrukt, s katerim je možno podpreti enako funkcionalnost.

Tako je za primer aktivnosti »Receive« možno pri modelu BPMN uporabiti konstrukt prejemanja opravil (*receive task*) oz. sporočilni dogodek (*message event*). Podobno velja za konstrukt »Wait«, ki se neposredno preslika v časovnik vmesnih dogodkov (*timer intermediate event*). Aktivnost »Exit« jezika BPEL je v modelu BPMN predstavljena prav tako z uporabo konstrukta dogodkov – zaključni dogodek (*termination end event*). Rezultat preslikav posameznih primitivnih aktivnosti jezika BPEL na model BPMN je prikazan v tabeli 3.1.

Aktivnost jezika BPEL	**Konstrukt notacije BPMN**	**Tip preslikave**
Invoke	sending/receiving task, message event	Delna
Receive	receiving task, message event	Neposredna
Reply	sending task, message event	Delna
Validate	-	Ni izvedljiva
Assign	assignment	Delna
Wait	timer intermediate event	Neposredna
Exist	termination end event	Neposredna
Throw, Rethrow	error end event	Delna
Compensate, CompensateScope	compensation events	Delna

Tabela 3.1: Prikaz uspešnosti preslikav posameznih primitivnih konstruktov jezika BPEL na konstrukte modela BPMN.

Primer nepopolne preslikave je aktivnost »Invoke«, ki v jeziku BPEL predstavlja točko dostopa do zunanjih storitev (preko partnerskih povezav). Izmenjava sporočil preko omenjene aktivnosti se razlikuje glede na način interakcije – asinhrona (enosmerna) ali sinhrona (zahteva – odgovor) komunikacija. V metamodelu BPMN abstrakcija partnerskih tipov in povezav ni podprta, a je kljub temu možno komunikacijo izvesti na nivoju spletnih storitev modela BPMN. Aktivnost »Invoke« je kljub temu lahko predstavljena znotraj notacije BPMN kot zaporedje dveh opravil – pošiljanje in prejemanje sporočil in vmesnih dogodkov (*sending/receiving task, message event*). Podobno velja za aktivnost »Reply«, kjer je sprejem sporočil (realiziran s sprejemanjem nalog – *sending task, message event*) sicer podprt, a vendar neposredna preslikava ni izvedljiva, saj konstrukt z enako funkcionalnostjo v modelu BPMN ne obstaja. Delno je pokrita preslikava tudi za eno od najbolj pogosto uporabljenih aktivnosti – »Assign«. Aktivnost predstavlja funkcionalnost prenašanja vrednosti spremenljivk iz spremenljivke A v spremenljivko B oz. kreiranje novih podatkov v primeru uporabe različnih izrazov (npr. matematičnih). Preslikava na model BPMN je podprta z uporabo konstrukta dodelitve (*assignment*). Težave se lahko pojavijo v primeru uporabe XPath izrazov, kjer enak izraz v jeziku BPEL ni zadovoljiv v modelu BPMN. Do podobne

situacije lahko pride tudi če uporabljamo bolj napreden način preoblikovanja podatkov (pri jeziku BPEL), saj je potrebno poskrbeti, da je enako preoblikovanje možno doseči z uporabo XPath izrazov v modelu BPMN. Jezik BPEL za obravnavo izjem znotraj določenega obsega funkcionalnosti uporablja aktivnosti »Throw«, »Rethrow« in tako imenovane upravljavce napak (*fault handlers*). Neposredna preslikava funkcionalnosti v enakem obsegu kot je izvedena v jeziku BPEL na model BPMN ni izvedljiva. Če je bilo izvajanje aktivnosti prekinjeno zaradi napake, uporabnik prejme obvestilo o napaki. Preslikava na vmesne napake znotraj določenega obsega aktivnosti, tako kot je to izvedeno pri jeziku BPEL, ni možna. V sklopu preslikave je pri modelu BPMN možna tudi vpeljava ekskluzivnega prehoda (*exclusive gateway*), ki poskrbi za ohranjanje procesne strukture, kot je definirana v jeziku BPEL. Delna preslikava na model BPMN je podprta z uporabo konstrukta končne napake dogodkov (*error end event*). Enako velja tudi za aktivnosti »Compensate« in »CompensateScope«, ki podpirajo funkcionalnost kompenzacije. Te aktivnosti so protislovje upravljavcem kompenzacij (*compensation handlers*) pri modelu BPMN. Do razhajanj prihaja predvsem v vrstnem redu izvedbe posameznih aktivnosti v sklopu izvajanja aktivnosti kompenzacije. Pri jeziku BPEL se izvajanje drugih aktivnosti, ki so definirane za aktivnostjo kompenzacije, izvede zaporedno, medtem ko se pri modelu BPMN pripadajoča opravila izvajajo vzporedno z aktivnostjo kompenzacije [41].

Skupino kompleksnih aktivnosti jezika BPEL predstavljajo aktivnosti:

- »Sequence«,
- »If-elseif-else«,
- »While«,
- »RepeatUntil«,
- »ForEach«,
- »Pick«,
- »Flow«,
- »Control links«,
- »Scope«
- »Fault handlers«,
- »Event handlers«,
- »Termination handler«,
- »Compensation handler«.

Večino naštetih kompleksnih aktivnosti jezika BPEL je možno neposredno preslikati v različne konstrukte modela BPMN razen določenih izjem, kjer je preslikava le delno izvedljiva – »Flow«, »Scope« in »Fault handlers«. Nekatere aktivnosti, ki po funkcionalnosti predstavljajo upravljavce dogodkov – »Event handler« in »Termination handler«, ne moremo preslikati na model BPMN, saj določeni koncepti delovanja upravljavcev niso podprti.

Neposredna preslikava konstruktov jezika BPEL je izvedljiva za aktivnost »Sequence«, saj obstaja podoben konstrukt – zaporedja toka (*sequence flow*), ki podpira funkcionalnost zaporedja tudi na modelu BPMN. Enako velja za aktivnost »If-elseif-else«, ki predstavlja funkcionalnost pogojne vejitve in je v modelu BPMN predstavljena kot ekskluzivni prehod (*exclusive data-based gateway*). Aktivnosti »While« in »RepeatUntil« sta prav tako podprti v modelu BPMN v obliki standardne aktivnosti ponavljanja (*standard loop activity*). Za aktivnost »ForEach« obstaja drugačen konstrukt na modelu BPMN, primerek večkratne aktivnosti pojavljanja (*multiple-instance loop activity*). Aktivnost »Pick«, ki predstavlja funkcionalnost dogodka, ki temelji na prejemanju sporočila, se na modelu BPMN preslika v prehod, ki temelji na dogodku (*event based gateway*). Upravljavec kompenzacij – aktivnost »Compensation handler« se na modelu BPMN preslika v kompenzacijsko aktivnost (*compensation activity*) oz. kompenzacijski dogodek (*compensation event*). Rezultat preslikav posameznih naprednih aktivnosti jezika BPEL na model BPMN je prikazan v tabeli 3.2.

Aktivnost jezika BPEL	Konstrukt notacije BPMN	Tip preslikave
Sequence	sequence flow	Neposredna
If-elseif-else	exclusive gateway	Neposredna
While, RepeatUntil	standard loop activity	Neposredna
ForEach	multiple-instance loop activity	Neposredna
Pick	event-based gateway	Neposredna
Flow, Control links	inclusive gateway, complex gateway	Delna
Scope	embedded sub process	Delna
Fault handler	exception flow	Delna
Event handler	-	Ni izvedljiva
Termination handler	-	Ni izvedljiva
Compensation handler	compensation event	Neposredna

Tabela 3.2: Prikaz uspešnosti preslikav posameznih primitivnih konstruktov jezika BPEL na konstrukte modela BPMN.

Primer nepopolne oz. delne preslikave predstavljata aktivnosti »Flow« in »Control links«, kjer pri preslikavi jezika BPEL v model BPMN naletimo na problem nedosegljive poti (*DPE – Death Path Elimination*). Omenjeni problem je rešljiv z uporabo dodatnih konstruktov na nivoju jezika BPEL – pogojnih prehodov in pogojnih stikov (*transition conditions*, *join conditions*). Pogojni prehodi se na modelu BPMN le delno preslikajo v vključujoče prehode (*inclusive gateway*). Enako velja tudi za pogojne stike, ki se prav tako ne morejo v celoti preslikati v kompleksne prehode (*complex gateway*). Posledično procese jezika BPEL, ki vsebujejo prej omenjene dodatne konstrukte, ne moremo uspešno preslikati predvsem zaradi nejasnih definicij tako vključujočih kot tudi kompleksnih prehodov na nivoju modela BPMN. Nepopolno preslikavo dosežemo tudi pri aktivnosti »Scope«, saj običajno vsebuje nabor

drugih aktivnosti, katerih doseg je omejen na izvajanje znotraj omenjene aktivnosti. Pri jeziku BPEL se ob vsaki pojavitvi aktivnosti »Scope« privzeto definirajo tudi upravljavci napak (*fault handler*), kompenzacij (*compensation handler*) in prekinitev (*termination handler*). Delegiranje dogodkov privzetih upravljavcev napak je skladno z modelom BPMN za razliko od upravljavcev kompenzacij in prekinitev, ki jih modeliramo eksplicitno na nivoju modela BPMN. Vzrok tega je povečanje kompleksnosti preslikanega modela BPMN. Podobno kot aktivnosti »Throw« in »Rethrow« je tudi preslikava upravljavca napak – aktivnost »Fault handler« na model BPMN le delno izvedljiva. Razlog zato leži v semantiki za delegiranje prekinitev, ki je pomanjkljivo definirana [13] pri modelu BPMN. Pomanjkljivost se nanaša predvsem na nejasno definicijo obravnavanja napak pri paralelnih instancah aktivnosti. Vsi omenjeni konstrukti so zaradi različnih tehničnih omejitev le delno podprti pri uporabi modela BPMN [41].

Skupino generičnih aktivnosti jezika BPEL tvorijo vsi preostali konstrukti, kot so:

- »Variables«,
- »Correlation mechanism«,
- »Process instantiation«,
- »Communication abstractions«.

Konstrukt »Variables« je eden od pomembnejših konstruktov pri jeziku BPEL, saj se uporablja pri skoraj vsaki aktivnosti. Njegova vloga je povezana z upravljanjem podatkov med procesom, kar je tudi iz samega imena razvidno. Podatki so shranjeni v spremenljivkah, ki so lahko glede na dosegljivost na nivoju samega procesa obravnavane kot globalne ali lokalne. Največkrat so spremenljivke definirane znotraj aktivnosti »Scope«, kjer so lokalno dosegljive. Spremenljivke uporabljamo tudi za shranjevanje podatkov, pridobljenih iz zunanjih virov (spletne storitve), za prenos vrednosti spremenljivk med aktivnostmi (preko uporabe »From« in »To« konstruktov) in tudi pri proženju napak. Model BPMN za upravljanje podatkovnih tokov uporablja koncept notacije podatkovnih objektov. Za razliko od jezika BPEL pri podatkovnih tokovih ne poznamo koncepta dosegljivosti spremenljivk znotraj aktivnosti. Preslikovanje spremenljivk z metodo uparjanja imen spremenljivk je tako posledično nesprejemljivo. Na podobno omejitev naletimo tudi pri proženju napak, saj pri modelu BPMN ni možno posredovati vrednosti spremenljivk, ker so le-te v času proženja napake nedosegljive [41]. Začetek izvajanja poslovnega procesa je v obeh modelih podprt z uporabo določenih aktivnosti, ki poleg vsega vsebujejo tudi konstrukt »Process instantiation« [12]. Na prvi pogled tako jezik BPEL kot tudi model BPMN podpirata podoben koncept in je preslikava med njima izvedljiva, a se izkaže, da pri določenih primerih to ni tako. Težava nastane, ko za pričetek izvajanja procesa ni zadolžena le ena sama aktivnost, ampak se proces lahko prične z več različnimi aktivnostmi. Pri jeziku BPEL imamo konkreten primer pri uporabi konstrukta »Flow«, kjer se aktivnosti izvajajo paralelno. Proženje nove instance

procesa je v tem primeru zakasnjeno, vse dokler se ne izvršijo dogodki znotraj konstrukta »Flow«. Zaradi pomanjkljivosti metamodela BPMN, kjer ni definiran atribut, ki podpira tak način izvedbe, takšnega primera ni možno preslikati [41]. Podobno kot ostali generični konstrukti sta tudi konstrukta »Correlation mechanism« in »Communication abstractions« le delno preslikana. Do razhajanj med jezikom BPEL in modelom BPMN pride zaradi različnih konceptov, pri izmenjavi sporočil in načinu komuniciranja z zunanjimi spletnimi storitvami. Rezultat preslikav posameznih generičnih aktivnosti jezika BPEL na model BPMN je prikazan v tabeli 3.3.

Aktivnost jezika BPEL	Konstrukt notacije BPMN	Tip preslikave
Variables	data object	Delna
Correlation mechanism	property	Delna
Process instantiation	message events	Delna
Communication abstractions	web service	Delna

Tabela 3.3: Prikaz uspešnosti preslikav posameznih primitivnih konstruktov jezika BPEL na konstrukte modela BPMN.

Za izvedbo zanesljive preslikave med omenjenima modeloma je potrebno odpraviti semantična neskladja in definirati dodatne razširitve modela BPMN, obenem pa ne smemo pretirano posegati v samo zasnovo, saj ta zagotavlja vzvratno združljivost. Pri postopku pretvorbe je treba zagotoviti preslikavo vseh podatkov, konstruktov in nastavitev iz jezika BPEL na model BPMN. V primeru, da pride do izgube določenih podatkov pri samem procesu, morajo le-ti biti dodani pri obratnem procesu kreiranja jezika BPEL iz modela BPMN. Pri tem so določeni podatki in procesne nastavitve lahko shranjeni tudi v atributih modela BPMN. Predpostavljamo, da je vsaka aktivnost jezika BPEL lahko predstavljena z enim ali več konstrukti notacije BPMN, primer take je konstrukt »Invoke« jezika BPEL, ki je preslikan v dva konstrukta modela BPMN – pošiljanje in prejemanje sporočil (*sending/receiving task, message event)*. A to ni vedno tako, saj so določene aktivnosti lahko preslikane neposredno med seboj, kot na primer aktivnosti »Receive« jezika BPEL, ki se preslika neposredno kot konstrukt prejemanja sporočila (*receiving task, message event)* pri modelu BPMN.

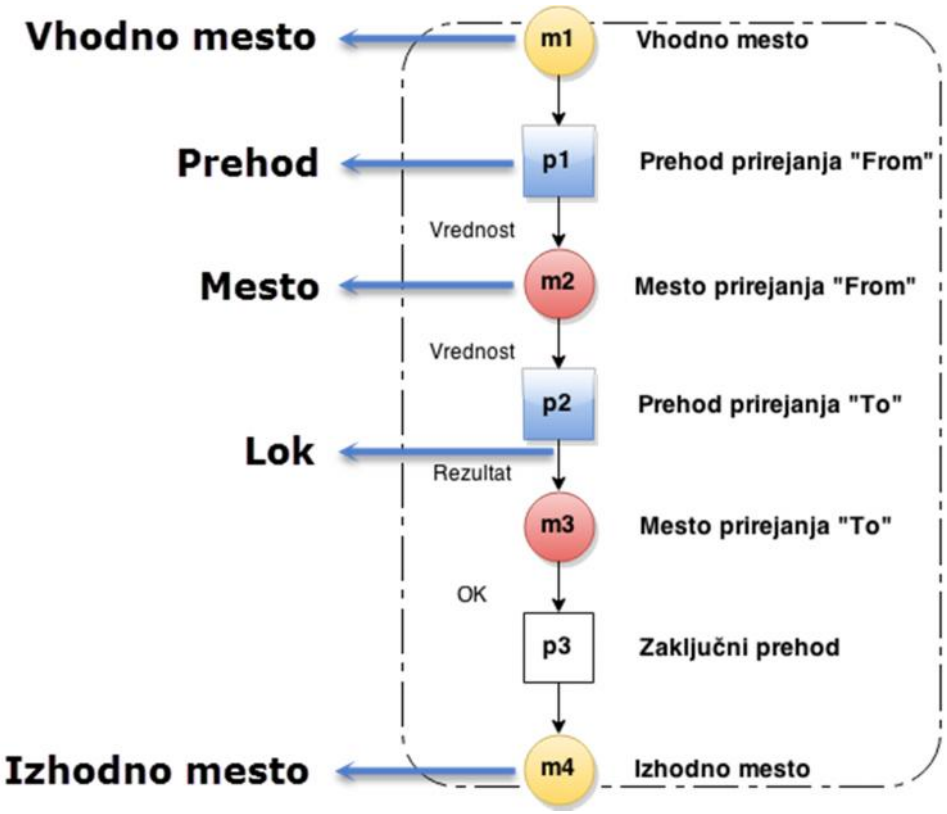
3.2 Preslikava jezika BPEL s pomočjo Petrijevih mrež

Pri preslikavi jezika BPEL se velikokrat osredotočimo le na osnovno logiko procesa in pri tem zanemarjamo druge še kako pomembne aspekte, kot so kontrolne povezave, problem eliminacije mrtve poti (DPE), zunanje (partnerske) povezave itd. Že dejstvo, da omenjeni aspekti spadajo med bolj napredne, samo po sebi pripomore k temu, da je področje v veliki meri še neraziskano. Ena od možnih različic preslikav je izdelava koncepta transformacije

jezika BPEL z uporabo razširjenih Petrijevih mrež, ki pripomore k formaliziranju opisa poslovnega procesa [44].

Petrijeve mreže predstavljajo temeljni mehanizem za modeliranje, formaliziranje, analiziranje in verifikacijo storitev programskih standardov in modelov. Zagotavljajo močan matematično podprt jezik za modeliranje, namenjen grafičnemu opisu paralelnih, asinhronih, distribuiranih sistemov [34]. Za opis modela je uporabljen enostaven grafični jezik z močno teoretično osnovo. Z njihovo uporabo lahko predstavimo vse konstrukte, uporabljene v obstoječem jeziku z odpravo tehnološko-orientiranih sintaktičnih omejitev (kot so npr. zanke) [25, 32]. Glavni razlog za široko uporabo Petrijevih mrež velja pripisati dejstvu, da so grafi predstavljeni v obliki grafične notacije in vsebujejo natančno definirano pomenoslovje, s katerim je možno izvajati formalne analize. Poleg tega omogoča enostaven prehod med konceptom nizko nivojskih mrež na visoko-nivojske, kar omogoča modeliranje tako primitivnih kot tudi bolj kompleksnih primerov [16].

Preproste Petrijeve mreže so predstavljene kot usmerjen graf, ki vsebuje dve vrsti vozlišč – prostore in prehode. Omenjena vozlišča so med seboj povezana s povezavami, ki so predstavljene kot loki, pri čemer povezava med dvema vozliščema enakega tipa ni možna. Primer Petrijeve mreže predstavlja slika 3.3. Pri izvedbi preslikav se uporabljajo preproste Petrijeve mreže (*place/transition nets*), sestavljene iz označenih in neoznačenih prehodov, z zajemanjem formalne semantike jezika BPEL. Označeni prehodi se uporabljajo pri modeliranju osnovnih aktivnosti (»Receive« in »Invoke«) in dogodkov, obenem pa vsebujejo tudi meta podatke, ki so povezani z osnovnimi aktivnostmi in dogodki (npr. imena tipov prejetih ali poslanih sporočil). Neoznačeni prehodi predstavljajo notranje akcije, ki niso opazne s strani zunanjih uporabnikov.



Slika 3.3: Primer aktivnosti »Assign« z uporabo konstruktov modela Petrijevih mrež.

Aktivnosti jezika BPEL so razdeljene v skupine glede na funkcionalnost. Poznamo tri glavne skupine aktivnosti: primitivne, kompleksne in generične. Aktivnosti, ki spadajo v posamezno skupino, so razložene v poglavju 3.1.1. Izvedljivost preslikav posameznih aktivnosti jezika BPEL na model Petrijevih mrež so prikazane v tabelah 3.4 in 3.5.

Aktivnost jezika BPEL	**Izvedljivost preslikave na model Petrijevih mrež**
Invoke	Izvedljiva
Receive	Izvedljiva
Reply	Izvedljiva
Validate	Izvedljiva
Assign	Izvedljiva
Wait	Izvedljiva
Exist	Izvedljiva
Throw, Rethrow	Izvedljiva
Compensate, CompensateScope	Izvedljiva

Tabela 3.4: Prikaz izvedljivosti preslikave primitivnih konstruktov jezika BPEL na model Petrijevih mrež.

V skupino primitivnih aktivnosti med drugim spadajo tudi komunikacijski aspekti, ki jih je možno modelirati s Petrijevimi mrežami. Pri jeziku BPEL so ti aspekti predstavljeni s tremi aktivnostmi – »Invoke«, »Receive« in »Reply«. Aktivnost »Invoke« se lahko izvaja v sinhronem (»zahteva – odgovor« – pošiljanje sporočila in čakanje na odgovora) ali asinhronem (enosmerna komunikacija – pošiljanje sporočila in ne čakanje odgovora) komunikacijskem načinu. Za modeliranje aktivnosti z uporabo Petrijevih mrež se uporabljata dva prehoda – »invoke_s« za pošiljanje sporočila in »invoke_r« za prejemanje sporočila. Mesto med prehodom »invoke_r« in »invoke_s« je modelirano kot vmesno stanje, kjer proces čaka na odgovor, ki je bil poslan drugemu procesu. Interakcija med dvema procesoma je modelirana v obliki dveh mest, imenovanih zahteva (*request*) in odziv (*response*). Mesto zahteva je zadolženo za posredovanje zahteve med procesoma. Vsebuje dva loka, kjer je prvi usmerjen s strani prehoda »invoke_s«, drugi pa je usmerjen proti prehodu za prejemanje (*receive*). Ravno nasprotno velja za drugi prehod – odziv, ki vsebuje vhodni lok s strani prehoda odgovor (*reply*) in izhodni lok proti prehodu »invoke_r« in je prav tako zadolžen za posredovanje odgovora med procesoma [28].

Podoben model kot velja za aktivnost »Invoke« velja tudi za preostale dve aktivnosti jezika BPEL, ki pokrivata komunikacijski aspekt – »Receive« in »Reply«. Pri aktivnosti »Receive« je z uporabo prehoda omogočeno čakanje na prihajajoče sporočilo, aktivnosti »Reply« pa je modelirana s prehodom, ki je namenjen pošiljanju odgovora na prej prejeto zahtevo (preko aktivnosti »Receive«). Modeliranje funkcionalnosti aktivnosti »Receive« je uporabljeno tudi pri upravljavcu dogodkov aktivnosti »Pick«, kjer se pri sporočilih tipa »onMessage« in »onAlarm« uporabi enaka implementacija. Med primitivne aktivnosti spadata tudi aktivnost »Validate« in »Assign«, ki predstavljata funkcionalnost ovrednotenja stanja spremenljivk in dodeljevanje vrednosti spremenljivkam. Z uporabo Petrijevih mrež sta modelirani na podoben način, le da aktivnost »Assign« obsega nekaj več prehodov in mest. Aktivnost »Assign« je sestavljena iz štirih mest in treh prehodov, ki so med seboj povezani z enosmernimi loki. Poleg začetnega ter končnega mesta vsebuje tudi dve vmesni mesti, ki predstavljata vmesni vrednosti na podlagi prehodov. Aktivnost »Validate« predstavlja okrnjeno različico aktivnosti »Assign« in je poleg vhodnega ter izhodnega mesta modelirana z dvema prehodoma in enim (vmesnim) mestom [21]. Podobno kot predhodni aktivnosti v sklopu preslikave konstrukta jezika BPEL modeliramo tudi aktivnost »Wait«, ki predstavlja primitivno aktivnost in sestoji iz začetnega ter končnega stanja in enega samega prehoda. Prekinitev celotnega procesa je izvedena z uporabo aktivnosti »Exit«. Ob proženju omenjene aktivnosti se morajo vse trenutno aktivne aktivnosti zaključiti v čim krajšem času brez upravljavcev napak in brez kompenzacije. Modeliranje aktivnosti »Exit« je izvedeno z uvedbo dodatnih zastavic – »no_exit« in »to_exit«, pri čemer je proces od začetka do konca izvajanja v stanju »no_exit«, razen v primeru pojavitve aktivnosti »Exit«. Takrat se stanje zastavice spremeni iz »no_exit« v »to_exit«. Za signalizacijo napak znotraj procesa uporabimo aktivnosti »Throw« in

»Rethrow«. Aktivnost »Throw« jezika BPEL lahko modeliramo kot implicitno ali eksplicitno, kjer pri implicitni obliki podamo tudi ime napake. Implicitna oblika je enostavna in je predstavljena samo z uporabo vhodnega mesta. Eksplicitna aktivnost »Throw« sestoji (poleg začetnega mesta in mesta napake) iz dodatnega mesta, ki vsebuje ime prehoda in napake. Za razliko od aktivnosti »Throw« je aktivnost »Rethrow« lahko uporabljena samo znotraj aktivnosti »Scope«. Modeliranje aktivnosti »Rethrow« z uporabo Petrijevih mrež se razlikuje le po dodatnem mestu, pri čemer je ime napake prebrano iz omenjenega dodatnega mesta definiranega s strani upravljavca napak. Podobno kot pri upravljavcu napak se pri upravljavcu kompenzacij jezika BPEL proži aktivnost tipa »Compensate«, ki poskrbi za kompenzacijo ene izmed aktivnosti oz. podaktivnosti, gnezdenih znotraj aktivnosti »Scope«. Modeliranje aktivnosti »Compensate« bo podrobneje opisano v nadaljevanju, v sklopu preslikave aktivnosti »Scope«.

Podobno kot modeliranje primitivnih aktivnosti poteka tudi preslikava kompleksnih aktivnosti. Izvedljivost preslikave kompleksnih aktivnosti jezika BPEL na model Petrijevih mrež je predstavljena v tabeli 3.5.

Aktivnost BPEL	**Izvedljivost preslikave na model Petrijevih mrež**
Sequence	Izvedljiva
If-elseif-else	Izvedljiva
While, RepeatUntil	Izvedljiva
ForEach	Izvedljiva
Pick	Izvedljiva z omejitvami
Flow, Control links	Izvedljiva
Scope	Izvedljiva
Fault handler	Izvedljiva pri aktivnosti Scope
Event handler	Izvedljiva pri aktivnosti Scope
Termination handler	Izvedljiva pri aktivnosti Scope
Compensation handler	Izvedljiva pri aktivnosti Scope

Tabela 3.5: Prikaz izvedljivosti preslikave kompleksnih konstruktov jezika BPEL na model Petrijevih mrež.

Ena od predstavnikov kompleksnih aktivnosti jezika BPEL je tudi aktivnost »Sequence«. Sestavljena je iz ene ali več aktivnosti, ki so izvršene zaporedno, za razliko od aktivnosti »Flow«, ki vpeljuje pojem paralelnosti in sinhronizacije aktivnosti. Obe aktivnosti sta modelirani na podoben način – z uporabo dveh prostorov, ki poskrbita za združitev na nivoju gnezdenih aktivnosti. Pri modeliranju aktivnosti »Flow« poleg dveh mest definiramo tudi dva prehoda, ki predstavljata začetek oz. konec – združitev gnezdenih aktivnosti. V obeh primerih lahko preslikavo obravnavamo kot neposredno preslikavo iz jezika BPEL na model Petrijevih

mrež. Aktivnost pogojne vejitve je predstavljena z aktivnostjo »If«, ki je na model Petrijevih mrež preslikana z uporabo oznak in dveh barv (*TRUE* in *FALSE*), saj aktivnost vsebuje pogoj, ki ga je potrebno pravilno ovrednotiti. Vrednosti barv označuje posamezen lok, ki predstavlja izvedbo določene poti (veje aktivnosti). Izbira poti je odvisna od ovrednotenja pogoja in primerjanja rezultata z možnimi barvami na prehodu [44].

Aktivnosti »While« in »RepeatUntil« jezika BPEL vsebujeta strukturirane zanke, saj sta sestavljeni iz pogoja in niza aktivnosti, ki se izvajajo, vse dokler je pogoj izpolnjen. Veljavnost pogoja je preverjena ob vsaki iteraciji; če ta ni izpolnjen, se izvajanje zanke prekine. Preverjanje pri teh dveh aktivnostih poteka različno: pri aktivnosti »While« ponavljanje zanke izvajamo, medtem ko je pogoj izpolnjen, pri aktivnosti »RepeatUntil« pa, dokler je pogoj izpolnjen. Obe aktivnosti sta na model Petrijevih mrež preslikani z uporabo začetnega prehoda, kjer preberemo vrednost spremenljivke, ki se nato preveri v pogoju, modeliranem z dodatnim mestom. Rezultat preverjanja pogoja določi izvedbo enega od prehodov (*TRUE* ali *FALSE*), ki se na podlagi tega izvede. Mesta in prehodi so med seboj povezani z usmerjenimi loki. Aktivnost »RepeatUntil« prav tako kot predhodno omenjeni aktivnosti predstavlja strukturirano zanko, a je za razliko od ostalih dveh izvedba možna tako v zaporednem kot tudi v vzporednem načinu. Modeliranje aktivnosti v zaporednem načinu izvajanja z uporabo Petrijevih mrež je enaka modeliranju aktivnosti »While« ali »RepeatUntil«, a je pri tem potrebno dodati še aktivnosti »Scope« in »Assign«. Modeliranje vzporednega načina preslikave na model Petrijevih mrež je izveden z omejenim številom instanc aktivnosti »Scope«, kjer modeliramo le instance te aktivnosti, ki predstavlja gnezden element. Modeliranje instance »Scope« bo opisano v nadaljevanju [21].

Aktivnost tipa »Pick« jezika BPEL predstavlja pogojno obnašanje, pri čemer je določena pogojna aktivnost izvršena na podlagi proženja zunanjega dogodka. »Pick« aktivnost vsebuje niz pogojnih aktivnosti, kjer je vzbujena tista aktivnost, katere dogodek je bil prožen. Poznamo dva tipa dogodkov – sporočilni dogodki (*onMessage*), ki se izvršijo ob prihodu zunanjega sporočila, in alarmni dogodki (*onAlarm*), ki se izvršijo ob prekinitvah (*timeout*). Modeliranje aktivnosti »Pick« je izvedeno z uporabo dveh različnih tipov prehodov glede na tip dogodkov. Število prehodov je odvisno od števila dogodkov, vsak dogodek je modeliran na enak način, kot so modelirane gnezdene aktivnosti pri konstruktu »Sequence«. Za razliko od proženja dogodka tipa »OnAlarm« je pri dogodku tipa »OnMessage« možno sočasno proženje več instanc. To lahko privede do težave, in sicer do neomejene mreže (*unbounded net*), pri čemer je analiza takega problema, gledano z računskega vidika, kompleksna. Omenjenemu problemu se lahko izognemo z uporabo podomrežij, kjer za vsako sproženo instanco upravljavca dogodkov uporabimo »n« dodatnih podvojenih podomrežij, ki poskrbijo za lovljenje »n« aktivnih instanc sočasno [28].

Aktivnost »Control links« velja za nestrukturiran konstrukt, ki predstavlja sinhronizirane odvisnosti med posameznimi aktivnostmi. V primeru, da določena aktivnost vsebuje izvorno povezavo, je ta v sklopu preslikave na Petrijeve mreže modelirana s prehodom izvorne povezave aktivnosti (*source link activity*), ki je postavljen za aktivnostjo. Za vsako izvorno povezavo aktivnosti moramo določiti prostor, ki predstavlja izhod omenjenega prehoda. Enako velja za aktivnost, ki vsebuje ponorno povezavo. V tem primeru je nov prehod ponorna povezava aktivnosti (*target link transition*), dodana pred aktivnostjo. Ponovno določimo mesto, ki pa v tem primeru predstavlja vhod za definiran prehod. Stanje kontrolne povezave je definirano z zastavico, ki vsebuje tri stanja: »TRUE«, »FALSE«, »UNSET« [8].

Preslikava aktivnosti »Scope« na model Petrijeve mreže je razdeljena na štiri dele glede na modeliranje različnih aspektov:

- pozitiven kontrolni tok, ki vsebuje gnezdene aktivnosti in upravljavce dogodkov,
- negativen kontrolni tok,
- upravljavce kompenzacij,
- upravljavce prekinitev.

Pozitiven krmilni tok je modeliran s prehodom inicializacije (*initialize*), ki definira stanje aktivnosti »Active«. To ostane nespremenjeno, vse dokler se ne izvedejo gnezdene aktivnosti ali upravljavci dogodkov. Ob končanju prehod dokončanja (*finalize*) ponastavi stanje aktivnosti na »!Active«, s čimer označi tok kot neaktiven, ter »Successful«, kar pomeni, da se je kontrolni tok končal brez napak. Negativen kontrolni tok je sestavljen iz upravljavcev napak in gnezdenih aktivnosti. V primeru pojavitve napake v gnezdeni aktivnosti se aktivira mesto, imenovano »fault_in«, obenem pa pri tem pride tudi do spremembe stanja aktivnosti iz aktivnega na neaktivno in proženja upravljavca napak. Pri tem se posreduje ime sprožene napake in prične z izvajanjem le-te. Če je slednje uspešno izvedeno, se označi mesto »final« in izvajanje aktivnosti »Scope« je zaključeno. V nasprotnem primeru se aktivira dodatno mesto »fault up«, ki delegira napako na nivo nadrejene aktivnosti oz. na nivo samega procesa. Modeliranje upravljavca kompenzacij je izvedeno s pomočjo treh prehodov – »pass«, »handle« in »skip«. Izvedba kompenzacije je sprožena s strani aktivnosti »Compensate« in »CompensateScope«, ki aktivirata mesto »compensate«. V primeru uspešne izvedbe gnezdene aktivnosti se sproži izvajanje njenega kompenzacijskega upravljavca, v nasprotnem primeru se izvajanje kompenzacije preskoči. Upravljavec prekinitev se izvede samo v primeru prekinitve gnezdene aktivnosti (mesto *inner stopped*) in če je pri tem ni prišlo do napake (mesto *active*) oziroma ni bila izvedena aktivnost »Exit« (mesto *!Exiting*). V tem primeru je sprožen prehod (»start th«), ki izvede upravljavce prekinitev, v nasprotnem primeru je označeno mesto (»stopped«) [21].

Generične aktivnosti jezika BPEL niso neposredno preslikane na model Petrijevih mrež, saj se posamezne aktivnosti (»Variables«, »Correlation set«) ne obravnavajo na enakem nivoju.

Konstrukti, kot sta »Correlation set« in »Process instantiation«, niso podprti s semantiko in ne pokrivajo celotnega življenjskega cikla kreiranja instanc procesa.

Jezik BPEL uživa veliko podporo pri procesno usmerjenih jezikih, kljub temu pa še vedno pridejo do izraza določene pomanjkljivosti, kot so: nedosegljivost določenih aktivnosti, slaba optimizacija na področju dohodnih sporočil itd. [28]. Omenjene slabosti lahko odpravimo s preslikavo jezika BPEL v Petrijeve mreže in uporabo obstoječih analitičnih tehnik. Preslikava je popolna z vidika konstruktov, ki podpirajo nadzor toka podatkov in komunikacijskih akcij. Obenem pa preslikava nudi tudi analizo dostopnosti določenih aktivnosti in s tem pripomore pri odpravi omenjenih pomanjkljivosti jezika BPEL.

3.2.1 Petrijeve odprte mreže delovnega toka – Open Workflow Nets (OWFN)

Poseben razred Petrijevih mrež predstavljajo tako imenovane odprte mreže (*OWF – Open Workflow Nets*), ki so predstavljene kot posplošena različica mrež delovnega toka. Odprte mreže delovnega toka OWFN predstavljajo nadgradnjo klasičnih Petrijevih mrež z vmesnikom procesa, ki je predstavljen kot niz vhodnih oz. izhodnih prostorov [1]. Dejansko gre za klasične Petrijeve mreže z različnimi zaporedji prostorov, ki so predstavljeni kot vmesniki okolja. Posledice interakcije med mrežo in okoljem so vidne v obliki žetonov, ki se prosto pojavljajo na odprtih prostorih. Odprti prostori lahko predstavljajo tako vhodno, kot tudi izhodno mesto oz. v določenih primerih tudi oboje, kjer se lahko poljubno pojavljajo žetoni s strani okolja [15]. Pri pretvorbi jezika BPEL v odprte mreže OWFN se uporablja klasičen pristop, kjer se vsak konstrukt jezika BPEL neposredno preslika v obliko odprtih mrež OWFN. Na ta način dobimo nabor vzorcev za posamezen konstrukt, pri čemer vsak od vzorcev vsebuje vmesnik za povezovanje z drugim vzorcem konstrukta. Zbirka med seboj povezanih vzorcev tako tvori semantiko jezika BPEL, ki je končna, saj pokriva vse klasične in izredne dogodke procesa kot tudi arbitrarna gnezdena področja in ponavljajoče se konstrukte. Postopek preslikave jezika BPEL v Petrijevo mrežo tipa OWFN se prične s preslikavo in kreiranjem abstraktnega sintaktičnega drevesa – AST (*Abstract Syntax Tree*), ki vsebuje podatke o sintaksi in strukturi procesa, aktivnostih in instancah. Sledi kreiranje grafa kontrolnih tokov – CFG (*Control Flow Graph*) na podlagi procesa, s katerim predstavimo izvajanje procesa po vseh možnih poteh ter izpeljemo hierarhično strukturo in identificiramo medsebojno povezane odvisnosti. Graf predstavlja delno zaporedje aktivnosti, kjer določimo tudi povezavo odvisnosti med posameznimi aktivnostmi. Na podlagi zaporedja lahko lažje določimo postopek kompenzacije znotraj določenega obsega (aktivnosti »Scope«). Sledi izločanje nemogočih poti – deli procesa, ki se nikoli ne bodo izvedli kar omogoča optimizacijo samega procesa in obenem zmanjša kompleksnost pretvorbe. Postopek preslikave nadaljujemo s prilagajanjem že omenjenega modela AST na obstoječe vzorce konstrukta. V ta namen imamo izdelano skladišče konstruktov, ki za vsako aktivnost vsebuje več različnih vzorcev. Na podlagi vseh zbranih informacij iz modela AST in vseh opomb iz

skladišča izberemo vzorec, ki se najbolje prilega konstruktu, ki ga trenutno obravnavamo [20]. Rezultat postopka je preslikan proces, sestavljen iz vseh izbranih vzorcev.

3.2.2 Petrijeve mreže delovnega toka – Workflow Nets (WFN)

Petrijeve mreže, ki modelirajo dimenzijo nadzora toka znotraj delovnega toka, predstavljajo poseben razred, imenovan mreže delovnega toka (*WFN – Workflow Nets*). So eden od standardov za modeliranje in analizo delovnih tokov. Večinoma se uporabljajo za abstrakcijo delovnega toka in preverjanje tako imenovane lastnosti smotrnosti oz. trdnosti poslovnega procesa. Omenjena lastnost zagotavlja (podobno kot pri OWF [43]), da v določenem delovnem toku ne more priti do zastojev in drugih anomalij, ki so lahko zaznani brez podrobnejšega poznavanja domene [2].

Pri mrežah WFN prehodi ustrezajo aktivnostim, kjer nekatere od aktivnosti predstavljajo »prave« aktivnosti, medtem ko so druge dodane v namen usmerjanja. Za razliko od klasičnih Petrijevih mrež WFN mreže vsebujejo en izvoren prostor, imenovan vhod, in en ponoren prostor, imenovan izhod, kjer za vsako vozlišče ali prehod obstaja povezava od izvora do ponora [3]. Mreže WFN predstavljajo generično osnovno različico Petrijevih mrež, na podlagi katerih so izdelane tudi odprte mreže OWF. Ključno razliko med mrežami WFN in OWFN predstavljajo nabori odprtih prostorov, ki predstavljajo vmesnike spletnih storitev.

Jezik BPEL predstavlja standard za modeliranje poslovnih procesov, kljub temu pa posveča premalo pozornosti verifikaciji poslovnega procesa. Posledično prihaja do težav, kot so nezmožnost zaznavanja zastoja, prepoznavanje delov procesa, ki so nedosegljivi itd. Pri uporabi jezika BPEL je močno poudarjena funkcionalnost povezovanja z zunanjimi storitvami, vendar preverjanje o uspešnosti izvedbe klica ni podprto. Podobna težava se lahko pojavi ob uspešnem zaključku procesa, kjer ni evidentno, ali smo uspešno opravili vse potrebne naloge, ne da bi katera od aktivnosti ostala nedokončana. V teh primerih pride v poštev preslikava jezika BPEL na mreže WFN, ki preko raznih verifikacijskih tehnik in orodij omogoča zaznavanje anomalij. Kot pri ostalih modelih tudi pri mrežah WFN obstajajo določene slabosti, kjer preslikava določenih aktivnosti (»Scope«, upravljavci napak) še ni dodobra podprta, a kljub temu predstavljajo dober koncept za verifikacijo poslovnih procesov [37].

4 Zasnova modela za pretvorbo jezika BPEL v delovne tokove spletne storitve SWF

Obstaja množica različnih načinov in tehnik, kako uporabiti, prilagoditi in preslikati koncept BPEL v nek drug, za potrebe reševanja določenega problema, bolj primeren model. V poglavjih 3.1 in 3.2 smo podali nekaj možnih rešitev preslikav, ki glede na zahteve in potrebe odpravljajo določene pomanjkljivosti. Preslikave niso popolne, ker se osredotočajo na reševanje določenega problema in ne pokrivajo celotnega spektra. Velikokrat so takšne delne rešitve sicer primerne za uporabo na določenem področju in pri reševanju specifičnih primerov. Na takšnih primerih se učimo in lahko ugotovimo prednosti oziroma slabosti določene preslikave. Lahko jih tudi uporabimo kot koncept pri načrtovanju novih preslikav, kjer bomo podprli reševanje takšnih problemov. V našem primeru smo se osredotočili predvsem na izvedbo preslikave vseh relevantnih konstruktov jezika BPEL v okviru potrditve koncepta preslikave na delovne tokove spletne storitve SWF. Predhodno obravnavane preslikave se podobno kot naš koncept osredotočajo na odpravo posameznih problemov, a nobena ne omogoča izvedbe poslovnega procesa na računalniškem oblaku v obliki delovnih tokov spletne storitve SWF.

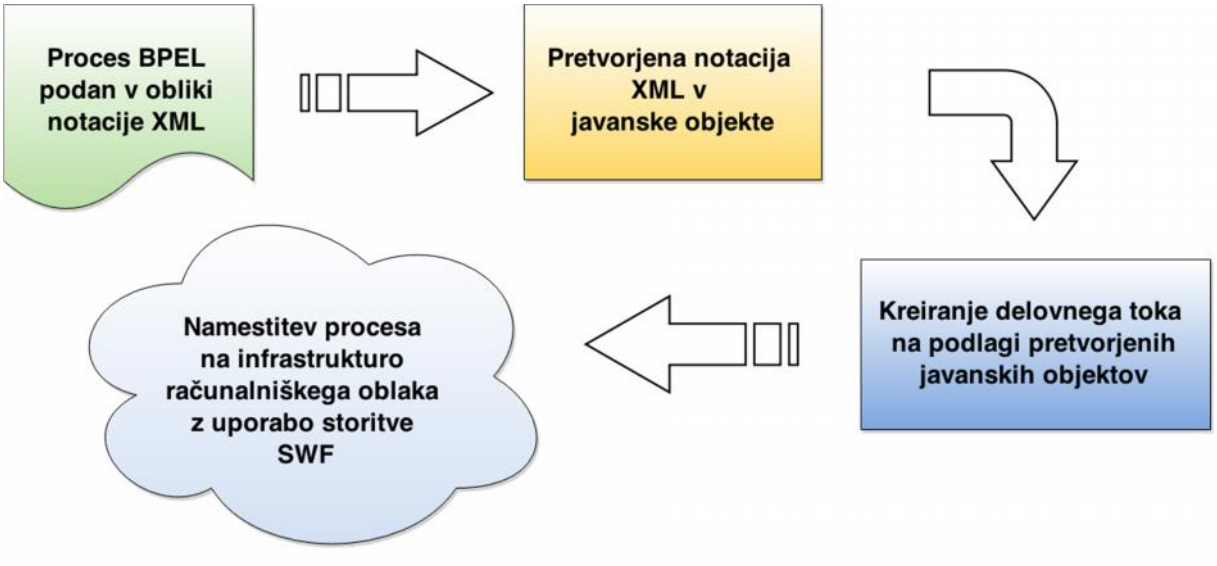
V našem primeru smo za izhodišče vzeli jezik BPEL, ki predstavlja standard za izvajanje poslovnih procesov, a ga ni možno izvajati neposredno na oblaku. V ta namen smo izdelali preslikavo jezika BPEL na delovne tokove spletne storitve SWF ter s tem omogočili izvajanje poslovnih procesov jezika BPEL na računalniškem oblaku (v sklopu storitve SWF). Spletna storitev SWF dejansko predstavlja premik v načinu razmišljanja in drugačen pristop k izvajanju aktivnosti, zato v našem primeru ne gre za klasično preslikavo med različnimi modeli. Analizirali smo pristope in načine delovanja delovnih tokov spletne storitve SWF ter na podlagi pridobljenih rezultatov analize izdelali model za preslikavo konstruktov jezika BPEL, da se slednji kar najbolje prilagodi storitvi računalniškega oblaka. Izdelani model smo podprli tudi z izdelavo prototipne aplikacije, kjer smo preizkusili izvajanje poslovnih procesov na oblaku z uporabo delovnih tokov spletne storitve SWF.

4.1 Izdelava modela

4.1.1 Koncept preslikave modela

Pri izdelavi modela smo si za cilj zadali izdelavo preslikave, kot je prikazana na sliki 4.1, kjer smo želeli obstoječ proces jezika BPEL preslikati na delovne tokove spletne storitve SWF. S tem smo želeli iz že obstoječega jezika BPEL pridobiti vse potrebne informacije, s katerimi lahko kreiramo nov »model«, ki bo primeren za izvajanje v okviru omenjene oblačne storitve. Pri načrtovanju rešitve modela smo upoštevali vse tri skupine aktivnosti (enostavne, napredne in generične), omenjene v poglavju 3.1.1, vendar pri implementaciji nismo pokrili vseh

aktivnosti iz posameznih skupin v celoti, saj smo se v tej fazi izdelave posvetili predvsem potrditvi samega koncepta.



Slika 4.1: Koncept preslikave poslovnih procesov jezika BPEL na delovne tokove spletne storitve SWF.

Vsaka aktivnost je v jeziku BPEL definirana z določenimi lastnostmi in meta podatki, ki so predstavljeni s pomočjo notacije XML. Predstavitev oz. zapis vsake aktivnost lahko torej vsebuje več različnih podatkov, ki se med seboj lahko razlikujejo v vsebinskem in strukturnem pomenu. Raznolikost je spremenljiva od aktivnosti do aktivnosti, s tem pa je tudi raznolika njena kompleksnost. Rezultat raznolikosti je aktivnost, ki je lahko enostavna in lahko berljiva, posledično tudi enostavna za preslikavo. Po drugi strani pa lahko ta isti tip aktivnosti vsebuje določene lastnosti, ki povečajo njeno kompleksnost (gnezdenja, obsegi itd.), kjer so potrebni dodatni mehanizmi za uspešno preslikavo. Vse aktivnosti jezika BPEL so definirane s strani organizacije OASIS, ki je odgovorna za standardizacijo samega jezika. Definicija aktivnosti z vsemi potrebnimi lastnostmi je zapisana kot predloga XSD, ki predstavlja izhodišče za kreiranje poslovnih procesov z uporabo jezika BPEL. Predloga je dostopna na spletni strani organizacije OASIS, za njeno razumevanje pa je potrebno predznanje s področja notacije XML in predlog XSD. Omenjena predloga služi kot osnova, kjer so definirana vsa pravila, variacije in omejitve vsake aktivnosti jezika BPEL. Vsako modeliranje aktivnosti BPEL mora bil izdelano v skladu s predlogo, saj v nasprotnem primeru ni veljavno. Predloga služi kot »pravilnik«, ki smo ga tudi upoštevali pri izdelavi vseh testnih primerov poslovnih procesov. Slednje smo kasneje uporabili pri preslikavi za potrebe izvajanja delovnih tokov spletne storitve SWF.

Spletna storitev SWF je predstavljena kot storitev tipa PaaS, ki je namenjena izvajanju delovnih tokov. Delovni tok je sestavljen iz zaporedja definiranih operacij, imenovanih

aktivnosti. Vsaka izmed aktivnosti spletne storitve SWF predstavlja poljubno zaporedje operacij. Vrstni red izvajanja aktivnosti je predhodno definiran in obenem tudi odvisen od rezultatov posameznih predhodnih aktivnosti. Izvajanje delovnega toka preko spletne storitve SWF je pogojeno s predhodno registracijo slednjega (in njegovih aktivnosti). S tem je omogočeno izvajanje poslovnega procesa v obliki delovnega toka spletne storitve SWF. Izvajanje delovnih tokov se odvija v ločenih okoljih, imenovanih domene. Vsak delovni tok pripada vsaj eni domeni. Domene predstavljajo nekakšen izvajalni prostor, v okviru katerega se izvajajo različne instance določenega delovnega toka. Spletna storitev SWF omogoča izvajanje več različnih delovnih tokov sočasno. Z uporabo domen zagotovimo izoliranost posameznih tokov in njihovih aktivnosti. Za uporabo posameznih aktivnosti je tako kot pri delovnih tokovih potrebna predhodna registracija le-teh. Posamezno aktivnost lahko registriramo v več različicah, ki vsebujejo različne implementacije. Vsaka registrirana aktivnost je lahko uporabljena večkrat znotraj istega delovnega toka in v več delovnih tokovih znotraj posamezne domene. Podobno pravilo velja tudi za delovne tokove, saj so slednji prav tako lahko uporabljeni večkrat znotraj posamezne domene.

4.1.2 Pristopi k preslikavi aktivnosti

Aktivnosti delovnega toka so predstavljene v obliki zaporedja operacij, ki skupaj tvorijo definicijo funkcionalnosti določene aktivnosti. Za uspešno izvedbo določene aktivnosti delovnega toka spletne storitve SWF je treba obstoječe aktivnosti preslikati v okvirje, kot jih predpisuje storitev SWF [5]. Ena od glavnih prednosti uporabe delovnih tokov spletne storitve SWF ja njena odprtost v smislu definiranja aktivnosti. Definicija aktivnosti, njena struktura in operacije so povsem prilagodljive glede na potrebe posameznika. Uporaba aktivnosti je pogojena s predhodno registracijo, kjer definiramo njeno ime in nabor vhodnih podatkov. Da bi lažje podprli dinamično izvajanje delovnih tokov in zagotovili neprekinjeno integracijo, je omogočena registracija več aktivnosti z enakim imenom, pri čemer se aktivnosti med seboj razlikujejo le po številki različice. Posamezne aktivnosti jezika BPEL predstavljajo zaključeno celoto. Izvajanje aktivnosti v obliki konstrukta delovnega toka spletne storitve SWF je pogojeno z izvedbo preslikave, ki poskrbi za pravilno interpretacijo aktivnosti. Izvajanje operacij, ki skupaj predstavljajo aktivnost jezika BPEL, je treba podrobno preučiti. Na podlagi rezultatov raziskave sledi definiranje konstruktov, ki predstavljajo preslikane aktivnosti. Vsak tip aktivnosti je predstavljen kot en konstrukt, ki pokriva implementacijo določene aktivnosti. Pri kreiranju preslikav aktivnosti delimo slednje na primitivne in kompleksne, pri čemer velja pravilo, da za vsako primitivno aktivnost obstaja neposredna preslikava v določen konstrukt. Primeri konstruktov, ki so bili neposredno preslikani, so aktivnosti »Receive«, »Invoke«, »Reply« in »Assign«. Omenjene aktivnosti se med seboj ne prekrivajo in ne vsebujejo skupnih funkcionalnosti, zato so preslikane v posamezne ločene konstrukte. Pri kompleksnejših aktivnostih neposredna preslikava zaradi napredne vsebine ni izvedljiva. Take tipe aktivnosti lahko preslikamo na več načinov (dekompozicija v enostavne

– primitivne aktivnosti, preslikava po delih itd.). V našem primeru smo se odločili za hibridno rešitev.

Hibridni pristop je efektiven predvsem pri aktivnostih, ki vsebujejo gnezdene podaktivnosti. V teh primerih pri postopku analize odkrivamo morebitne vzorce primitivnih konstruktov. Gnezdene elemente nato identificiramo kot primitivne konstrukte, jih izluščimo in uporabimo njihove že preslikane konstrukte. Identificirane gnezdene elemente lahko izvajamo na več načinov glede na njihov obseg in odvisnost od hierarhične strukture. Gnezden element je lahko predstavljen tudi kot manjši delovni tok – podtok znotraj aktivnosti. Izvedba podtoka je tipično pogojena z veljavnostjo določenega pogoja. Tak primer predstavlja aktivnost »If«, kjer se izvedba veji na dva dela in se glede na veljavnost pogoja izvede le ena izmed vej. Izvedbo določene veje lahko interpretiramo kot izvedbo manjšega delovnega toka. Tak delovni tok lahko registriramo kot nov delovni tok, ki predstavlja hierarhično podrejen element – podtok. Rezultati izvedbe gnezdenega toka so ob zaključku izvajanja posredovani nadrejeni aktivnosti, ki jih predstavi kot svoje rezultate. V primerih, kjer je uporaba gnezdenega delovnega toka neprimerna, zaporedje gnezdenih aktivnosti izvedemo v obliki rekurzije. Rekurzija v tem primeru zamenja gnezden delovni tok (podtok), obenem pa zagotovi izvajanje vseh aktivnosti v pravilnem vrstnem redu ne glede na nivo gnezdenja. Princip rekurzije smo uporabili v primerih, kjer smo želeli preslikati pogojno ponavljajoče se segmente, saj je znano, da zanke lahko zapišemo tudi v obliki rekurzije. Določene napredne konstrukte zaradi svoje kompleksnosti nismo mogli preslikati na podlagi opisanih postopkov. Nekatere primere takih konstruktov (npr. »Fault handlers«) smo poskušali preslikati z uporabo konstruktov na nivoju programskega jezika (»try-catch« blok). Taka preslikava ni ravno najbolj primerna, saj je odvisna od programskega jezika in jo je težko nadzorovati oz. spreminjati v smislu prilagajanja implementacije na funkcionalnost aktivnosti. Vrstni red izvajanja aktivnosti je bil ne glede na način preslikave enak zaporedju izvajanja aktivnosti pri uporabi jezika BPEL. Seznam aktivnosti, ki jih vsebuje nek poslovni proces, smo na strani delovnega toka podali kot enega od vhodnih parametrov. Izvajanje posameznih aktivnosti smo izvedlo v skladu s seznamom aktivnosti. V primeru, da se je čas izvajanja določene aktivnosti ali gnezdenega delovnega toka povečal zaradi obsega ali kompleksnosti aktivnosti, to ni vplivalo na vrstni red izvajanja aktivnosti. Tak primer izvedbe, kjer delovni tok čaka na zaključek gnezdenega delovnega toka, smo podprli pri uporabi aktivnosti »While«. Podobno velja pri aktivnosti, ki za svoje izvajanje potrebuje podatke iz zunanjih spletnih storitev. V ta namen smo uporabili funkcionalnost aktivnosti »Invoke«, ki na podlagi vhodnih podatkov izvede zahtevo na zunanjo spletno storitev. Zahteva je izvedena v sinhronem načinu, kar pomeni, da omenjena aktivnost čaka na odgovor. Izvajanje delovnega toka je v tem primeru začasno zaustavljeno, dokler ne prejmemo odgovora zunanje spletne storitve.

Princip izvajanja delovnih tokov in aktivnosti pri uporabi spletne storitve SWF temelji na asinhroni komunikaciji, saj se vsaka posamezna aktivnost obravnava kot ločena zaključena celota. Poslovni procesi so za razliko od spletne storitve SWF predstavljeni kot zaporedje aktivnosti, ki skupaj tvorijo celoto. Izvajanje aktivnosti je na strani spletne storitve SWF treba prilagoditi, da bo predstavljena kot manjši del celote. Razhod med obema pristopoma je možno prikriti z uporabo določenih povezovalnih členov (npr. skupnih spremenljivk, vhodnih podatkov), ki v našem primeru predstavljajo vhodne podatke posamezne preslikane aktivnosti. Ti poskrbijo za prehode med aktivnostmi in na tak način med seboj povežejo zaporedne aktivnosti v celoto – poslovni proces. Naloga tako imenovanih povezovalnih členov je hranjenje podatkov o trenutno izvedeni aktivnosti in posredovanje le-teh novi aktivnosti (v podanem zaporedju) v obliki vhodnih parametrov, s pomočjo katerih se začne izvajanje naslednje aktivnosti. Na ta način premostimo oviro med dvema aktivnostma in ju med seboj povežemo. Na podlagi zaporedja izvajanja aktivnosti med seboj povežemo vse konstrukte v podanem nizu in s tem tvorimo poslovni proces. Rezultati posameznih aktivnosti se med seboj prenašajo in dopolnjujejo med samim izvajanjem. Končni rezultat poslovnega procesa je predstavljen kot skupni rezultat izvajanja posameznih aktivnosti.

4.2 Analiza in preslikava aktivnosti jezika BPEL v delovne tokove spletne storitve SWF

Tako, kot je vsak proces jezika BPEL sestavljen iz več aktivnosti, so tudi delovni tokovi sestavljeni iz posameznih konstruktov, ki skupaj tvorijo celoto. Pri postopku analize smo podrobno preučili vsako izmed aktivnosti jezika BPEL in vse njene variacije. Na podlagi rezultatov smo identificirali posamezne ključne dele ter tako poskušali definirati okvir implementacije za konstrukt delovnih tokov spletne storitve SWF. Posamezne aktivnosti smo analizirali po skupinah, omenjenih v poglavju 3.1.1. Preslikava posamezne aktivnosti je bila odvisna od skupine, pa tudi od same kompleksnosti, kar je privedlo do raznolikosti preslikav glede na funkcionalnost posamezne aktivnosti. Tako smo določene aktivnosti, ki so predstavljale enostavno aktivnost jezika BPEL, identificirali kot kompleksen konstrukt (»Assign«) delovnih tokov spletne storitve SWF. Določene aktivnosti, ki so pripadale skupini kompleksnih aktivnosti (»While«, »RepeatUntil«, »ForEach«) in so vsebovale pogojno ponavljajoče se cikle, smo identificirali kot podtok oz. ponavljanje ciklov v obliki rekurzije. Vseh aktivnosti nismo neposredno identificirali kot konstrukte delovnih tokov spletne storitve SWF, ampak smo njihovo funkcionalnost podprli v obliki tako imenovanih povezovalnih členov – nastavitvenih aktivnosti (»Import«, »PartnerLink«, »Variables«), ki so bili v uporabi med celotnim postopkom izvedbe delovnega toka.

4.2.1 Nastavitvene aktivnosti

4.2.1.1 Opis nastavitvenih aktivnosti

Nastavitvene aktivnosti in njihove lastnosti so v uporabi med celotnim poslovnim procesom. Znano je, da vsak poslovni proces sestavljen iz dveh delov – poslovne logike, ki predstavlja dejansko vsebino poslovnega procesa, in pripadajoče spletne storitve, ki predstavlja vstopno točko poslovnega procesa. Pripadajoča spletna storitev se obnaša kot odjemalec in je zadolžena za proženje instance poslovnega procesa in vnos vhodnih podatkov. Za povezavo med pripadajočo spletno storitvijo – odjemalcem in poslovnim procesom skrbi konstrukt »Import«. Njegova naloga je povezovanje dveh med seboj ločenih dokumentov na podlagi podatkov o lokaciji, imenskem prostoru in tipu dokumenta. Z uporabo omenjenega konstrukta lahko med seboj povezujemo različne datoteke, a gre v večini primerov za povezavo med zunanjimi predlogami XSD ali, kot v našem primeru, za povezavo z zunanjimi – pripadajočimi spletnimi storitvami. Z vzpostavitvijo povezave lahko posamezne aktivnosti poslovnega procesa dostopajo do podatkov, ki so definirani na strani odjemalca. Na tak način sta med seboj povezana tudi odjemalec in aktivnost »Receive«, ki je podrobneje opisana v poglavju 4.2.3. Uporaba povezanih podatkov pride najbolj do izraza pri uporabi spremenljivk, definiranih znotraj konstrukta »Variables«. Z njegovo uporabo definiramo vse spremenljivke, ki jih potrebujemo za izvedbo poslovnega procesa. Posamezne spremenljivke poslovnega procesa so definirane z uporabo konstrukta »Variable« znotraj konstrukta »Variables«. Vsaka spremenljivka je sestavljena iz imena in tipa. Odjemalci običajno za zagon poslovnega procesa posredujejo tudi vhodne podatke, pridobljene s strani uporabnika. Rezultat izvedbe poslovnega procesa je običajno predstavljen v obliki nekakšnih izhodnih podatkov, ki so nato kot odgovor posredovani odjemalcu. Da bi podprli vhodne in izhodne podatke, vsak poslovni proces definira vsaj dve spremenljivki, ki sta predstavljeni s konstruktom »Variable«, pri čemer je tip spremenljivk enak tipu vhodnih oz. izhodnih podatkov odjemalca. Z drugimi besedami povedano, struktura vhodne oz. izhodne spremenljivke je definirana s strani odjemalca preko konstrukta »Import«, ki poskrbi za uvoz definicij spremenljivk. Poslovni proces uporablja za komunikacijo z zunanjimi spletnimi storitvami in tudi odjemalcem konstrukt »PartnerLinks«. Gre za nastavitveni konstrukt, ki je definiran z imenom in tipom povezave ter vlogo. Podobno kot pri drugih aktivnostih, tudi tu do podatkov o imenu in tipu zunanje spletne storitve dostopamo preko konstrukta »Import«. Primer poslovnega procesa jezika BPEL z uporabo omenjenih konstruktov »Import«, »PartnerLinks« in »Variables« prikazuje slika 4.2.

```
<!-- Konstrukt import in odjemalec WSDL -->
<bpel:import location="WS_CurrencyConverterArtifacts.wsdl"
             namespace="http://my.ws.converter.service/"
             importType="http://schemas.xmlsoap.org/wsdl/" />
<!-- ================================================================= -->
<!-- PARTNERLINKS                                                      -->
<!-- Seznam storitev, ki sodelujejo v procesu BPEL.                    -->
<!-- ================================================================= -->
<bpel:partnerLinks>
      <!-- The 'client' role represents the requester of this service. -->
            <bpel:partnerLink name="client"
                  partnerLinkType="tns:WS_CurrencyConverter"
                  myRole="WS_CurrencyConverterProvider"/>
            <bpel:partnerLink name="CurrencyConverterPL"
                  partnerLinkType="tns:CurrencyConverterPLT"
                  partnerRole="CurrencyConverterPLRole"/>
</bpel:partnerLinks>
<!-- ================================================================= -->
<!-- VARIABLES                                                         -->
<!-- Seznam spremenljivk, ki bodo uporabljeni v procesu BPEL           -->
<!-- ================================================================= -->
<bpel:variables>
      <!-- Vhodno sporočilo s strani pripadajočega odjemalca -->
      <bpel:variable name="input"
            messageType="tns:WS_CurrencyConverterRequestMessage"/>
       <!-- Izhodno spočilo, ki bo posredovano pripadajočem odjemalcu -->
      <bpel:variable name="output"
            messageType="tns:WS_CurrencyConverterResponseMessage"/>
      <bpel:variable name="CurrencyConverterPLResponse"
            messageType="tns:getConvertedCurrencyResponse"/>
       <bpel:variable name="CurrencyConverterPLRequest"
            messageType="tns:getConvertedCurrencyRequest"/>
</bpel:variables>
```

Slika 4.2: Nastavitveni konstrukti »Import«, »PartnerLinks« in »Variables« poslovnega procesa jezika BPEL na primeru spletne menjalnice.

4.2.1.2 Preslikava nastavitvenih aktivnosti

Delovni tok, ki predstavlja poslovni proces, za svoje pravilno delovanje potrebuje vnaprej določene vhodne parametre. Vsebina vhodnih podatkov je v našem primeru sestavljena iz več delov. Povezava med konstrukti »Import«, »PartnerLinks« in »Variables« je opisana v prejšnjem poglavju 4.2.1 in se uporablja med celotnim poslovnim procesom, enako pa velja tudi za povezave teh konstruktov z drugimi aktivnosti. V ta namen nastavitvenih aktivnosti nismo preslikoval neposredno. Vsako izmed tako imenovanih nastavitveno-povezovalnih aktivnosti smo predstavili kot nabor vhodnih spremenljivk delovnega toka. Poleg tega so bili omenjeni vhodni podatki posredovani tudi vsaki izmed aktivnosti. Na ta način smo zagotovili, da ima vsaka aktivnost dostop do nastavitveno-povezovalnih podatkov, potrebnih za nemoteno izvajanje delovnega procesa.

Na podlagi konstrukta »Import« in definicije spremenljivk, ki so bile v uporabi preko konstrukta »Variables«, smo pridobili potrebne informacije o definiciji in strukturi vhodnih

ter izhodnih podatkov s strani pripadajočega odjemalca. Ti podatki so predstavljali osnovo za definiranje spremenljivk in njihovih struktur, uporabljenih med poslovnim procesom. Pridobljene podatke smo preslikali v dve ločeni podatkovni strukturi, ki hranita informacije o vhodnih oz. izhodnih podatkih. Vsebina podatkovnih struktur se je razlikovala, saj ena izmed podatkovnih struktur hrani podatke o imenu spremenljivke in njeni definiciji, druga pa vsebuje dejanske vrednosti določenih spremenljivk. Poleg podatkov o spremenljivkah smo na podoben način preoblikovali tudi podatke o aktivnosti »PartnerLinks«. V tem primeru smo kot vhodni podatek delovnega toka podali podatkovno strukturo, ki vsebuje podatke o zunanji spletni storitvi, ki je povezana preko aktivnosti »PartnerLinks«. Dodaten parameter v vhodnih podatkih delovnega toka je predstavljal tudi sam poslovni proces. Ta je definiran v obliki seznama aktivnosti, ki jih poslovni proces vsebuje. Vrstni red aktivnosti, posredovanih delovnemu toku spletne storitve SWF, je enak vrstnemu redu aktivnosti, definiranih v poslovnem procesu jezika BPEL. Z uporabo opisanih vhodnih parametrov smo registrirali delovni tok in podali vse potrebne podatke za začetek izvajanja poslovnega procesa v obliki delovnega toka.

4.2.2 Aktivnost Sequence

4.2.2.1 Opis aktivnosti Sequence

Poslovna logika oz. logika orkestracije poslovnega procesa je običajno zajeta kot vsebina aktivnosti »Sequence«. Slednja vsebuje nabor drugih aktivnosti, ki so izvršene v zaporednem vrstnem redu. Kot prikazuje slika 4.3, je možno gnezditi znotraj konstrukta »Sequence« vse druge aktivnosti, tudi aktivnost »Sequence«. Omenjena aktivnost sicer spada v skupino kompleksnih aktivnosti. Ta narašča s številom in tipom elementov drugih aktivnosti, ki so gnezdene znotraj aktivnosti »Sequence«. Običajno je logika poslovnega procesa sestavljena tako, da je na prvo mesto postavljen konstrukt, ki prejme podatke s strani odjemalca in sproži začetek izvajanja poslovnega procesa. Sledi izvajanje določenih aktivnosti, ki predstavljajo jedro poslovnega procesa in so definirane znotraj aktivnosti »Sequence«. Slednja predstavlja glavni – očetovski element, ki vsebuje seznam aktivnosti, ki bodo izveden v zaporedju. Zaključek poslovnega procesa je predstavljen v obliki posredovanja pridobljenega rezultata pripadajočemu odjemalcu.

Slika 4.3: Primer aktivnosti "Sequence" in gnezdenje različnih drugih aktivnosti jezika BPEL.

4.2.2.2 Preslikava aktivnosti Sequence

Aktivnosti poslovnega procesa, ki predstavljajo poslovno logiko procesa, so, kot smo videli v prejšnjem poglavju, združene v obliki zaporedja gnezdenih podaktivnosti aktivnosti »Sequence«. Naloga omenjenega konstrukta je izvajanje posameznih aktivnosti v zaporedju glede na položaj gnezdenih aktivnosti. V našem primeru smo aktivnost glede na njen položaj preslikali na dva načina. Prvi način je namenjen preslikavi aktivnosti kot strukture, ki vsebuje operacije – aktivnosti poslovnega procesa in predstavlja glavni element samega procesa. V tem primeru smo omenjeno aktivnost preslikali v obliki niza gnezdenih aktivnosti, ki si sledijo v zaporedju. Zaporedni niz aktivnosti smo posredovali delovnemu toku v obliki enega od vhodnih parametrov. Drugi način preslikave smo uporabili ob primerih gnezdenja omenjene aktivnosti znotraj drugih konstruktov. V teh primerih smo aktivnost obravnavali kot kompleksno in posledično smo tudi njeno preslikavo opravili na drugačen način. Primer take preslikave je bolj podrobno opisan v nadaljevanju pri preslikavah aktivnosti »If« (poglavje 4.2.5.1) in »RepeatUntil« (poglavje 4.2.6.1).

4.2.3 Aktivnosti Receive, Reply in Invoke

4.2.3.1 Opis aktivnosti Receive, Reply in Invoke

Za proženje nove instance poslovnega procesa je zadolžena aktivnost »Receive«. Slednja predstavlja stičišče med poslovnim procesom in odjemalcem. Na nek način predstavlja poslušalca dogodkov pripadajočega odjemalca. Sestavljena je iz imena aktivnosti in nekaj dodatnih podatkov o samem odjemalcu, kot so: ime partnerske povezave, tip vrat in operacije odjemalca ter ime spremenljivke. Slednje je povezano s predhodno obravnavanim konstruktom »Variable«, ki predstavlja definicijo spremenljivke. Na podlagi njenega imena zagotovimo, da lahko poslovni proces preko odjemalca prejme vhodne podatke in jih hrani v pravilni obliki. Podatki, ki so shranjeni v obliki spremenljivke, so običajno uporabljeni v poslovnem procesu. Dodatni parameter, ki definira aktivnost »Receive«, je podatek o kreiranju nove instance. Ta podatek predstavlja obnašanje poslovnega procesa ob proženju pripadajočega odjemalca. Parameter definira potrebo po proženju nove instance poslovnega procesa, če ta še ne obstaja. S tem dejanjem je sprožen začetek izvajanja poslovnega procesa,

čemur sledi izvajanje ostalih aktivnosti, ki predstavljajo jedro (poslovno logiko) samega procesa.

Tako kot konstrukt »Receive« predstavlja začetno aktivnost, konstrukt »Reply« predstavlja končno aktivnost poslovnega procesa. Slednji prav tako spada v skupino primitivnih aktivnosti in se uporablja pri sinhronem načinu izvedbe poslovnega procesa. V tem primeru konstrukt »Receive« predstavlja zahtevo, konstrukt »Reply« pa odgovor. Funkcionalnost konstrukta je predstavljena kot odgovor izvedbe poslovnega procesa in je podobno kot konstrukt »Receive« definirana s podatki o tipu vrat ter operaciji pripadajočega odjemalca, s katero je povezan preko uporabe konstrukta »PartnertLinks«. Aktivnosti »Receive« in »Reply« sta med seboj povezani in vsebujeta enake vrednosti prej omenjenih parametrov. Razlikujeta se le v vsebini podatkov, shranjenih v obliki vrednosti spremenljivke, ki bo posredovana pripadajočemu odjemalcu. V primeru, da pri izvajanju poslovnega procesa pride do napake, je aktivnost »Reply« sposobna (namesto odgovora) posredovati sporočilo o napaki pripadajočemu odjemalcu. Oba omenjena konstrukta s svojimi lastnostmi sta predstavljena na sliki 4.4.

Slika 4.4: Primera konstruktov "Receive" in "Reply", ki vsebujeta dodatne podatke o pripadajočem odjemalcu.

Tipi aktivnosti, pa tudi njihovo zaporedje, so odvisni od poslovne logike. Njeno izvajanje pogosto zahteva tudi povezovanje z zunanjimi spletnimi storitvami. Za potrebe povezovanja uporabljamo aktivnost »Invoke«. Ta preko podatkov, definiranih v konstruktu »PartnerLink« in imenu partnerske povezave, omogoči dostop do zunanje spletne storitve. Podobno kot aktivnosti »Receive« in »Reply« tudi aktivnost »Invoke« vsebuje podatke o tipu vrat in operaciji zunanje spletne storitve. Poleg tega pa vsebuje tudi podatke o tako imenovani vhodni in izhodni spremenljivki (Slika 4.5). Vhodna spremenljivka definira strukturo podatkov, ki so predvideni s strani zunanje spletne storitve in predstavljajo vhodne podatke – zahtevo. Izhodna spremenljivka definira strukturo, ki predstavlja odgovor in je s strani zunanje spletne storitve posredovana aktivnosti »Invoke« – odgovor. Pri povezovanju na zunanje spletne storitve pomembno vlogo igrajo predhodno omenjeni nastavitveni konstrukti

(poglavje 4.2.1), ki skrbijo za izvajanje potrebnih klicev na nižjih plasteh. Izvajanje zahtev na zunanjih spletnih storitvah ločimo na sinhrone in asinhrone. Pri sinhronih klicih ob izvedbi klica čakamo na pripadajoč odgovor, medtem ko pri asinhronem klicu ne čakamo odgovora in nadaljujemo z izvajanjem ostalih aktivnosti glede na definiran vrstni red.


```
<bpel:invoke name="KlicStoritveSpletneMenjalnice"
       partnerLink="CurrencyConverterPL"
       operation="getConvertedCurrency"
       portType="tns:CurrencyConverterServicePortType"
       inputVariable="CurrencyConverterPLRequest"
       outputVariable="CurrencyConverterPLResponse">
</bpel:invoke>
```

Slika 4.5: Aktivnost "Invoke" z vsemi potrebnimi podatki za izvedbo klica zunanje spletne storitve.

4.2.3.2 Preslikava aktivnosti Receive, Reply in Invoke

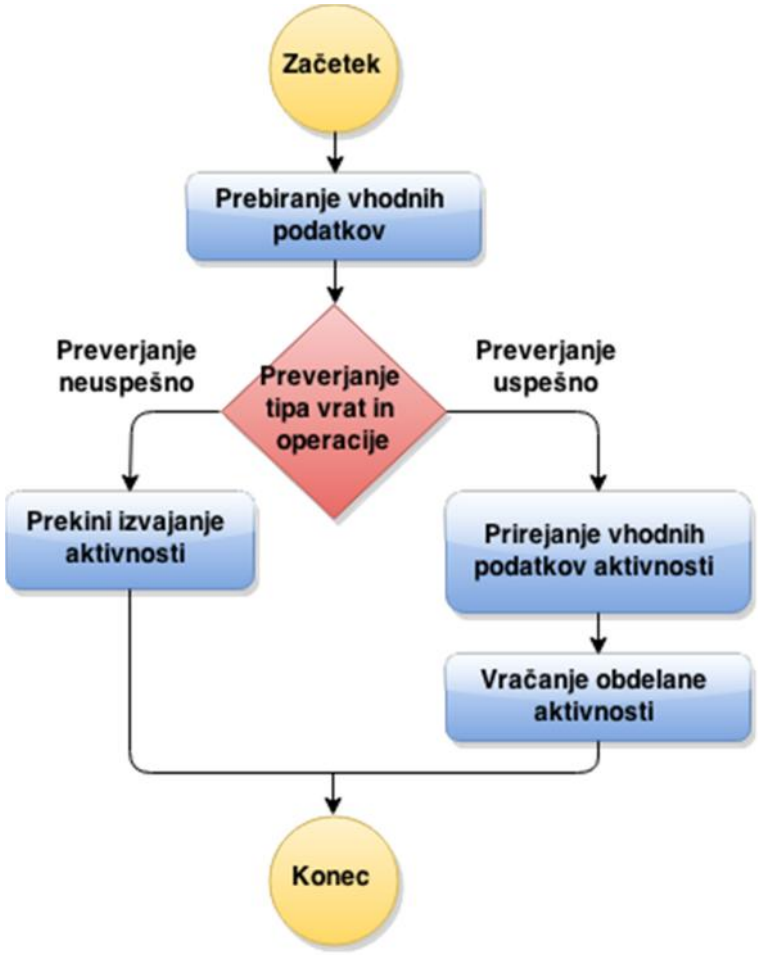
Zagon poslovnega procesa se običajno sproži preko aktivnosti »Receive«, prav tako pa je možen tudi s proženjem aktivnosti »Pick«, kar je predstavljeno v razdelku 4.2.7. Aktivnost »Receive« smo neposredno preslikali v konstrukt delovnega toka spletne storitve SWF, kot prikazuje slika 4.6. Konstrukt smo preslikali kot primitivno aktivnost, ki na podlagi podanih podatkov o tipu vrat in operaciji identificira pripadajočega odjemalca ter sprejme vhodne podatke s strani pripadajočega odjemalca. Na podlagi identifikacije odjemalca smo opravili prirejanje vhodnih podatkov, prispelih s strani odjemalca, na podatkovne strukture delovnega toka. Na ta način smo vpeljali vhodne podatke v poslovni proces.

```
@Asynchronous
private Promise<TActivity> processActivity(Promise<TActivity> activity,
                     Promise<ActivityTypeEnum> activityType,
                     @Wait Promise<Map<String, Object>> variablesHolder,
                     @Wait Promise<Map<String, Object>> variableValuesHolder,
                     @Wait Promise<Map<String, Object>> additionalOptions) {
      Promise<List<TActivity>> sequenceActivityList;
      ActivityTypeEnum aType = activityType.get();
      switch (aType) {
            case RECEIVE:
                  return operations.processReceive(activity, variablesHolder,
                                                   variableValuesHolder);
            case REPLY:
                  return operations.processReply(activity, variablesHolder,
                                                   variableValuesHolder);
            case INVOKE:
                  return operations.processInvoke(activity, variablesHolder,
                                                   variableValuesHolder);
```

Slika 4.6: Preslikava aktivnosti »Receive«, »Reply« in »Invoke« v obliki klicev funkcije, ki vsebuje implementacijo posamezne aktivnosti.

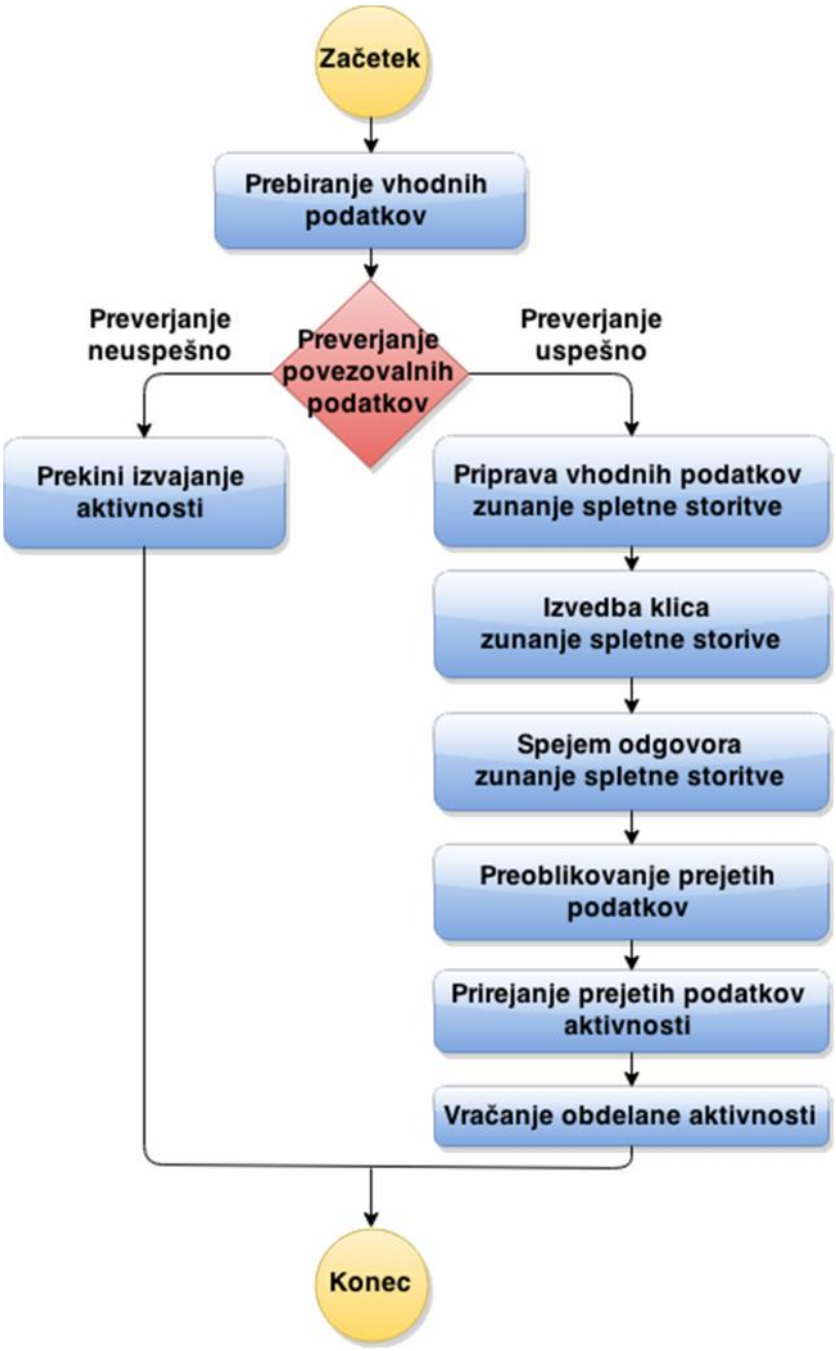
Aktivnosti »Receive« in »Reply« sta med seboj povezani, saj sodelujeta z enakim odjemalcem. Njuna funkcionalnost je prikazana v obliki diagrama na sliki 4.7. Medtem ko je aktivnost »Receive« zadolžena za sprejemanje podatkov, je funkcionalnost aktivnosti »Reply« ravno obratna – zadolžena je za hranjenje odgovora, ki je kasneje posredovan odjemalcu.

Aktivnost »Reply« je podobno kot aktivnost »Receive« preslikana neposredno v konstrukt delovnega toka. V obeh primerih gre za podano funkcionalnost, zato se tudi pri aktivnosti »Reply« opravi preverjanje tipa vrat in operacije pripadajočega odjemalca. Na podlagi uspešne identifikacije se opravi prirejanje podatkov na določeno spremenljivko, ki je definirana s strani aktivnosti. Omenjena aktivnost deluje kot zaključna aktivnost delovnega toka, če je slednji izveden v sinhronem načinu. V nasprotnem primeru se delovni tok lahko konča tudi z drugim tipom aktivnosti. Prirejeni podatki so ob zaključku delovnega procesa posredovani pripadajočemu odjemalcu v obliki odgovora spletne storitve. Odjemalec nato poskrbi za posredovanje dobljenega odgovora uporabniku pripadajoče spletne storitve. Končnemu uporabniku so poslovni procesi in njihovo izvajanje predstavljeni v obliki klasične spletne storitve.



Slika 4.7: Diagram poteka aktivnosti »Receive« in »Reply«.

Za potrebe povezovanja z zunanjimi spletnimi storitvami smo aktivnost »Invoke« implementirali kot aktivnost, sestavljeno iz več korakov, kot prikazuje slika 4.8. Začeli smo z identifikacijo zunanje spletne storitve na podlagi vhodnih parametrov, ki so vsebovali vse potrebne podatke za uspešno izvedbo povezovanja z zunanjimi storitvami. Omenjene podatke smo pridobili s pomočjo konstruktov »Import« in »PartnerLinks«, ki nista bila preslikana kot klasična samostojna konstrukta v sklopu delovnega toka spletne storitve SWF. Njune podatke smo podali v obliki vhodnih parametrov delovnega toka. Na podlagi pridobljenih vhodnih podatkov in strukture slednjih smo pripravili veljaven format zahteve, ki je ustrezal definiciji zunanje spletne storitve. S pripravljeno zahtevo smo izvedli klic zunanje spletne storitve. Spletna storitev je posredovala odgovor na podlagi preddefinirane definicije, ki smo ga uspešno sprejeli. Prejet odgovor smo priredili in prilagodili glede na tip ter definicijo strukture, tako da se je odgovor prilegal podatkovnim strukturam, ki so predstavljale vhodne podatke vseh ostalih aktivnosti. S tem postopkom smo na strukturiran način zagotovili izvajanje klicev na poljubno zunanjo spletno storitev na osnovi podatkov (ki so predstavljali vhodne podatke delovnega toka), pridobljenih s strani procesa jezika BPEL.



Slika 4.8: Diagram poteka aktivnosti "Invoke".

4.2.4 Aktivnost Assign

4.2.4.1 Opis aktivnosti Assign

Ena najbolj uporabljanih aktivnosti, ki jo lahko najdemo v vsakem poslovnem procesu, je aktivnost »Assign«. Pripada skupini primitivnih aktivnosti, že ime samo po sebi predstavlja njeno funkcionalnost – dodeljevanje/prirejanje. V primeru poslovnih procesov to pomeni prirejanje vrednosti posameznim spremenljivkam. Struktura konstrukta lahko variira glede na konfiguracijo, s tem pa variira tudi sama kompleksnost. Konstrukt je sestavljen iz več manjših konstruktov, ki skupaj predstavljajo funkcionalnost prirejanja. Osnovni namen konstrukta je prirejanje vrednosti spremenljivk v smislu A = B oz. A = 5 itd. Imena in definicije

spremenljivk, ki jih uporabljamo za prirejanja, so definirane s konstruktom »Variable«. Funkcionalnost prirejanja vrednosti je razdeljena na dva dela: prvi je predstavljen kot izvorna vrednost ali izvorna spremenljivka, drugi pa kot ponorna spremenljivka oz. ponorna vrednost. V primeru uporabe spremenljivk le-te uporabljamo iz nabora spremenljivk konstrukta »Variables«. Za potrebe prirejanja je na voljo tudi uporaba izraznega jezika, ki z uporabo jezika za poizvedbe XPath pripomore pri luščenju podatkov, in sicer predvsem pri uporabi podatkov v obliki notacije XML. Struktura podatkov mora biti definirana v skladu z definicijami konstrukta »Variables«. Slika 4.9 prikazuje primer aktivnosti »Assign«, kjer je prikazana ena izmed možnih različic prirejanja vrednosti spremenljivk z uporabo notacije XML in jezika za poizvedbe XPath.

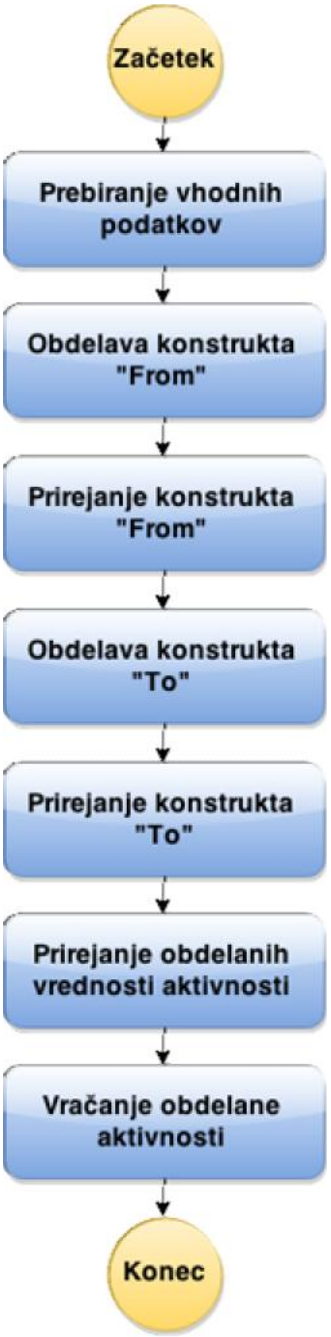
```
<bpel:assign validate="no" name="PrirejanjeVrednostiZaPretvarjanje">
  <bpel:copy>
    <bpel:from>
      <bpel:literal>
        <tns:getConvertedCurrencyRequest
             xmlns:tns="http://my.ws.converter.service/"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
          <tns:input>
            <tns:fromCurrency>tns:fromCurrency</tns:fromCurrency>
            <tns:toCurrency>tns:toCurrency</tns:toCurrency>
            <tns:fromAmount>0</tns:fromAmount>
          </tns:input>
        </tns:getConvertedCurrencyRequest>
      </bpel:literal>
    </bpel:from>
    <bpel:to variable="CurrencyConverterPLRequest" part="parameters"/>
  </bpel:copy>
  <bpel:copy>
    <bpel:from part="payload" variable="input">
      <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
        <![CDATA[tns:input]]>
      </bpel:query>
    </bpel:from>
    <bpel:to part="parameters" variable="CurrencyConverterPLRequest">
      <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
        <![CDATA[tns:input]]>
      </bpel:query>
    </bpel:to>
  </bpel:copy>
</bpel:assign>
```

Slika 4.9: Primer aktivnosti "Assign" s prirejanjem vrednosti XML določeni spremenljivki in vrednosti izluščenih podatkov XML z uporabo jezika za poizvedbe XPath.

4.2.4.2 Preslikava aktivnosti Assign

Za razliko od jezika BPEL, kjer aktivnost »Assign« sodi v skupino primitivnih aktivnosti, konstrukt »Assign« delovnega toka spletne storitve SWF ni predstavljal enostavne aktivnosti. Omenjeno aktivnosti lahko zaradi kompleksnosti in variacij pri opravljanju preslikave (Slika

4.10) prestavimo v skupino kompleksnih aktivnosti. Funkcionalnost omenjene aktivnosti sicer predstavlja primitivno operacijo prirejanja vrednosti podatkov, a je to prirejanje pogojeno s samo strukturo in definicijo podatkov. Raznolikost aktivnosti pride do izraza predvsem pri uporabi zunanjih spletnih storitev in prirejanju pridobljenih vrednosti, definiranih s strani spletne storitve. Znano je, da spletne storitve izmenjujejo podatke v obliki notacije XML, zato so bile tudi vsebine spremenljivk definirane v skladu s pripadajočimi shemami XSD spletne storitve. Takšna oblika podatkov je primerna tudi za uporabo pri drugih aktivnostih, predvsem tistih, ki so v svoji strukturi uporabljale pogoje in vejitve. Pri takih podatkih smo lahko na enostaven način izkoriščali pogoje v obliki izraznega jezika, ki je omogočal tudi uporabo jezika za poizvedbe XPath. V primerih, kjer podatki niso bili strukturirani v obliki notacije XML, je bilo potrebno predhodno opraviti pretvorbo. Na določene težave in omejitve smo naleteli pri uporabi primitivnih tipov podatkov, saj jih je bilo potrebno pretvarjati v strukturirane podatke v obliki notacije XML. Ta preslikava podatkov je včasih obojestranska in je odvisna od same kompleksnosti ter tipa spremenljivk v postopku prirejanja. Aktivnost »Assign« je ena od najbolj pogosto uporabljanih aktivnosti, saj jo najdemo tudi pri kompleksnih aktivnostih v obliki gnezdenih aktivnosti, zato je njena preslikava predstavljala velik zalogaj. Rezultati prirejanj so podobno kot pri ostalih aktivnostih shranjeni v obliki podatkov, ki so posredovani kot vhodni parametri pri vseh ostalih aktivnostih. Na ta način smo zagotovili dosegljivost ne glede na obliko in strukturo. Pri preslikavi strukturiranih podatkov smo uporabili različne tehnike in pristope, kot so bili uporabljeni pri primitivnih tipih. Zaradi svoje raznolikosti smo aktivnosti »Assign« preslikali v ločen konstrukt, ki je vseboval kompleksno implementacijo prirejanja podatkov. Pri implementaciji konstrukta smo se osredotočili na pokritje vseh različnih variacij prirejanj (spremenljivka – spremenljivka, primitivna vrednost – spremenljivka, strukturirana vrednost – spremenljivka …). Pri tem smo si pomagali z izdelavo pomožnih funkcij, ki so poskrbele samo za določene primere variacij prirejanj. Podobno velja za prirejanje vrednosti z uporabo jezika za poizvedbe, kjer smo podprli tudi prirejanja za osnovni nabor jezika za poizvedbe XPath.




```
Object toObject = null;
Object fromObject = null;

fromObject = handleAssign(contentValue,
                          fromCco,
                          activityVariableHolder,
                          activityVariableValuesHolder,
                          additionalOptions);
if(fromObject != null)
{
      toObject = handleAssign(fromObject,
                    toCco,activityVariableHolder,
                    activityVariableValuesHolder,
                    additionalOptions);
      contentValue = toObject;
}

//    prirejanje nove vrednosti seznama spremenljivk in
//    vrednosti
if(activityVariableHolder != null &&
   activityVariableValuesHolder != null)
{
      log.debug("Loading from activitiyVariables");
      variablesHolder = activityVariableHolder;
      variableValuesHolder = activityVariableValuesHolder;
}
if(contentValue == null) {
      log.error("An error occured assign content is
                 empty.");
      return null;
}
variableValuesHolder.put(toVariable, contentValue);
```

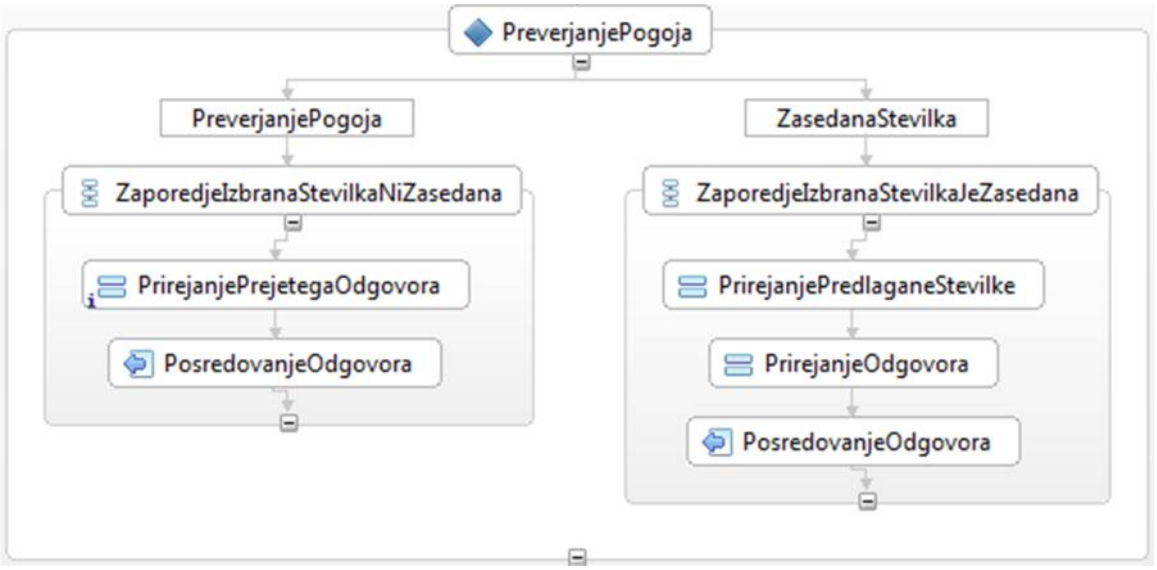
Slika 4.10: Diagram poteka aktivnosti »Assign« in izvleček programske kode, ki skrbi za prirejanje vrednosti med konstrukti »From« in »To«.

4.2.5 Aktivnost If

4.2.5.1 Opis aktivnosti If

Sestavljene aktivnosti podpirajo bolj kompleksno funkcionalnost in so predstavljene s kombinacijo primitivnih aktivnostih. Vsak kompleksen konstrukt za svoje izvajanje uporablja določen nabor primitivnih funkcionalnosti, kjer so primitivne aktivnosti velikokrat predstavljene v obliki gnezdenih elementov. Enako velja za konstrukt, ki predstavlja zaporedje izvajanja aktivnosti in je predstavljen z uporabo aktivnosti »Sequence«, ki lahko

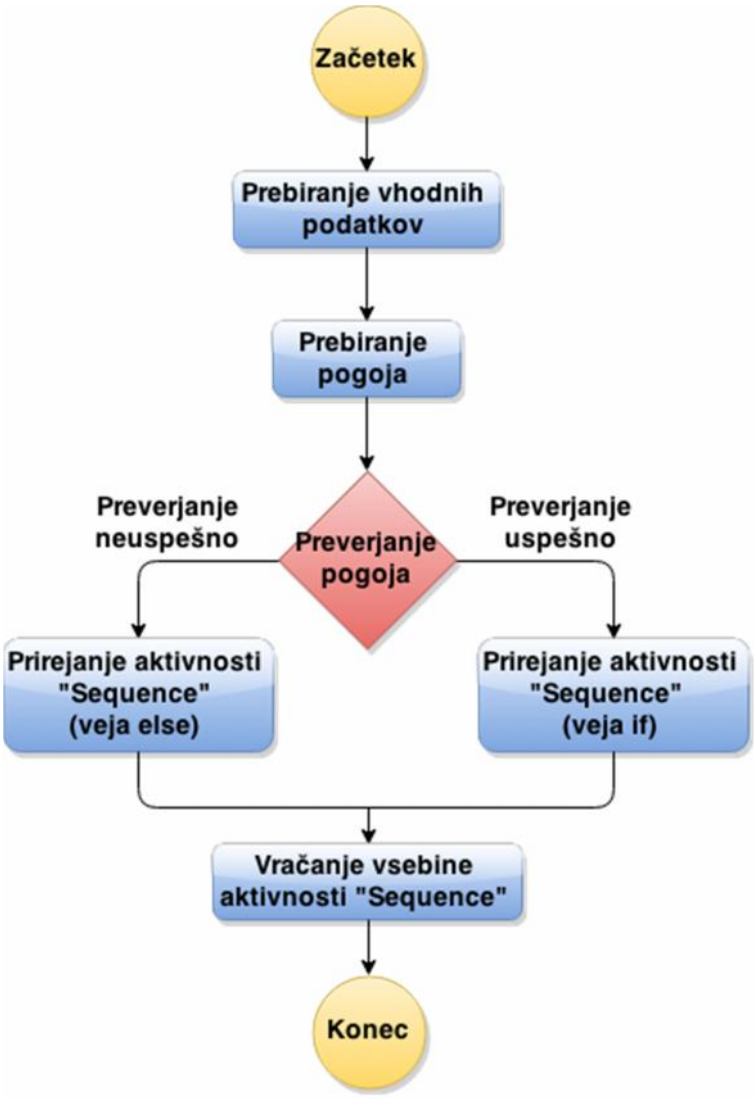
vsebuje vse ostale aktivnosti (v obliki gnezdenja aktivnosti) nekega poslovnega procesa. Njegova funkcionalnost je že opisana v poglavju 4.2.2. Znano je, da kompleksne aktivnosti vsebujejo primitivne konstrukte, vendar poleg teh lahko vsebujejo tudi druge kompleksne konstrukte. Primer take aktivnosti predstavlja pogojna aktivnost »If«, ki omogoča izvajanje določene aktivnosti glede na definiran pogoj. Aktivnost »If« je tako kot mnoge druge kompleksne aktivnosti sestavljena iz manjših konstruktov, ki skupaj tvorijo funkcionalnost pogojne vejitve. Ta običajno vsebuje dve ločeni veji, ki sta predstavljeni z dvema kompleksnima aktivnostma »Sequence«. Predstavljeni sta kot edina elementa vejitve in vsebujeta zaporedje gnezdenih aktivnosti. Na podlagi veljavnosti pogoja se izvede le ena izmed vej – aktivnosti »Sequence« (Slika 4.11). Pogoj je podan v obliki izraznega jezika, ki omogoča tudi uporabo jezika za poizvedbe XPath. Vsebina pogoja je odvisna od poslovne logike, obenem pa sam pogoj lahko vsebuje tudi vrednosti spremenljivk, ki so uporabljane v poslovnem procesu.



Slika 4.11: Aktivnosti "If" jezika BPEL na primeru preverjanja razpoložljivosti izbrane telefonske številke.

4.2.5.2 Preslikava aktivnosti If

Pri izvedbi preslikave kompleksne aktivnosti »If« je zaporedje gnezdenih aktivnosti razdeljeno na dve različni aktivnosti tipa »Sequence«, kjer se, kot prikazuje slika 4.12, glede na veljavnost pogoja izvrši le ena.



Slika 4.12: Diagram poteka aktivnosti "If", kjer se glede na izpolnjen pogoj izvrši ena od aktivnosti "Sequence".

Pri izvedbi konstrukta »If« na podlagi gnezdene aktivnosti »Sequence« pride do spremembe vrstnega reda izvajanja aktivnosti definiranega poslovnega procesa, zato je tudi ta aktivnost izvedena v skladu s prej opisano strategijo. Glede na veljavnost pogoja se namreč izvede zaporedje določenih gnezdenih aktivnosti, ki se izvajajo v obliki rekurzivnih klicev glede na globino gnezdenja. Ta je lahko poljubna, s tem pa se posledično spremeni vrstni red izvajanja aktivnosti, za kar poskrbijo konstrukti odločevalci. Aktivnost »If« tako preslikamo v aktivnost, ki preveri veljavnost pogoja in na podlagi tega odgovori s primerno vsebino v obliki aktivnosti »Sequence«. To zaporedje se na nivoju delovnega toka z uporabo rekurzivnih klicev izvede v definiranem vrstnem redu (Slika 4.13). Končni rezultat gnezdenih aktivnosti se nato priredi konstruktu »If«. Nadaljnje izvajanje aktivnosti, definiranih v delovnem toku, ki predstavlja poslovni proces, je zaustavljeno, vse dokler se ne zaključi izvajanje gnezdenih aktivnosti konstrukta »If«.

```
case IF:
      Promise<TSequence> sequence = operations.processIfElse(activity,
                                                variablesHolder,
                                                variableValuesHolder);
      sequenceActivityList = processSequence(sequence);
      return this.getProcessedActivity(sequenceActivityList,
                                       operations.getVariables(),
                                       perations.getVariableValues(),
                                       null);
```

Slika 4.13: Preslikava aktivnosti »If« v obliki klica funkcije, ki vsebuje implementacijo opisano z diagramom poteka.

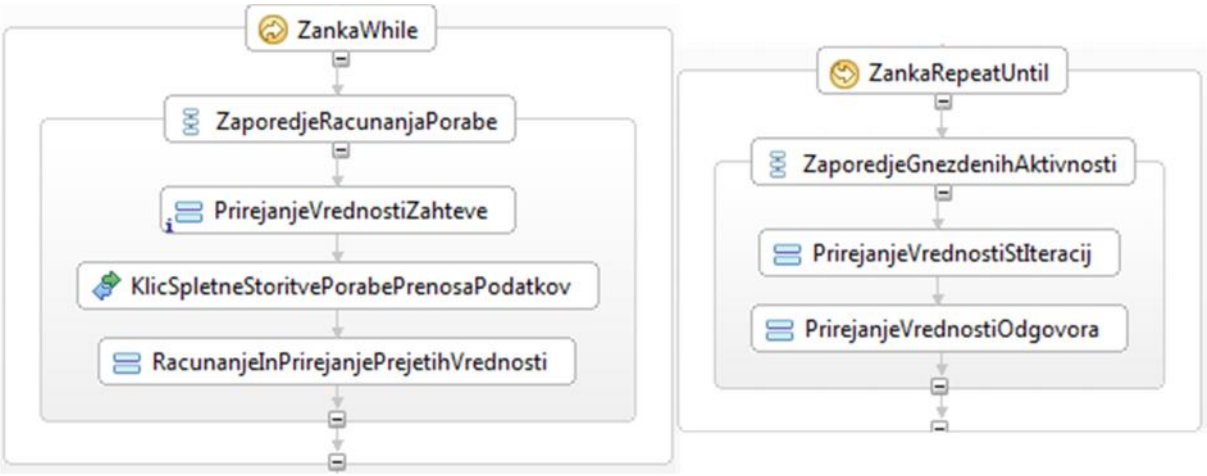
4.2.6 Aktivnosti While, RepeatUntil in ForEach

4.2.6.1 Opis aktivnosti While, RepeatUntil in ForEach

Pri pregledu aktivnosti »If« (poglavje 4.2.5) smo prikazali primer, kjer so znotraj aktivnosti gnezdeni tudi določeni kompleksni konstrukti. Podobno velja tudi za konstrukte, ki definirajo pogojno ponavljajoče se segmente aktivnosti. V tem primeru gre za konstrukte, ki so sestavljeni iz gnezdenih aktivnosti in pogoja, ki se uporablja tudi pri aktivnosti »If« in je namenjen definiciji pogoja vejitve – »Condition«. Pogoj je tako kot pri aktivnosti »If« podan v obliki izraznega jezika, ki omogoča uporabo jezika za poizvedbe XPath. Omenjen konstrukt je prav tako uporabljen pri vseh drugih aktivnostih, ki so predstavljene v obliki zank. Osnovno različico zank predstavlja aktivnost »While«, katere implementacijo lahko najdemo v veliko programskih jezikih današnjega časa. Gre za aktivnost, ki izvaja ponavljajoče se segmente kode v obliki iteracij na podlagi definiranega pogoja. Število ponavljajočih se iteracij je pogojeno z veljavnostjo pogoja, saj se ponavljanje preneha izvajati, ko pogoj ni več veljaven. Aktivnost »While« je torej sestavljena iz pripadajočega pogoja in gnezdenih aktivnosti, ki se izvajajo, vse dokler je pogoj veljaven. Veljavnost pogoja je običajno odvisna od vrednosti spremenljivke gnezdene aktivnosti, v nasprotnem primeru pride do tako imenovane mrtve zanke (*Dead lock*). To pomeni, da je število iteracij omejeno glede na vrednost spremenljivke, ki je vsebovana v eni izmed gnezdenih aktivnosti in se spreminja med vsako iteracijo, kar pa obenem vpliva tudi na veljavnost pogoja. Zato ne čudi dejstvo, da je pri uporabi aktivnosti »While« vedno gnezdena aktivnost »Assign«, ki je zadolžena za prirejanje vrednosti, vsebovane v pogoju. Poleg obveznega konstrukta »Assign«, ki skrbi za prirejanje, omenjena aktivnost lahko vsebuje tudi druge poljubne gnezdene aktivnosti.

Podobna funkcionalnost kot jo predstavlja aktivnost »While«, je uporabljena tudi pri aktivnosti »RepeatUntil«. Glavna razlika je predvsem v položaju pogoja, obenem pa aktivnost »RepeatUntil« bolj striktno določa zaporedje izvajanja gnezdenih aktivnosti z uporabo gnezdenega konstrukta »Sequence«. Funkcionalnost, izvedba pogoja, število iteracij in prirejanje vrednosti števca z uporabo gnezdene aktivnosti »Assign« so enaki kot pri aktivnosti

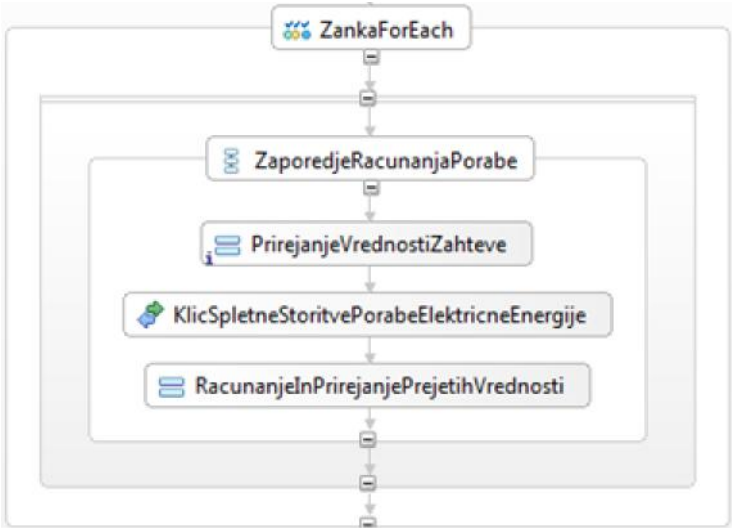
»While« z razliko mesta, kjer se pogoj preverja. Položaj pogoja je pomemben zaradi izvajanja števila iteracij, saj je v primeru aktivnosti »RepeatUntil« preverjanje veljavnosti pogoja prestavljeno na zadnje mesto. To pomeni, da bo nova iteracija izvršena, preden bo preverjena veljavnost pogoja, kar je ravno nasprotno kot pri aktivnosti »While«, kjer je pogoj preverjen, preden je izvedena iteracija in z njo vse gnezdene aktivnosti. Primer obeh aktivnosti prikazuje slika 4.14.



Slika 4.14: Aktivnosti "While" na primeru izračuna porabe prenosa podatkov in aktivnost "RepeatUntil" na testnem primeru, kjer se v odgovor zapiše vrednost števca iteracij.

Poleg dveh omenjenih aktivnosti, ki predstavljata funkcijo zank, v jeziku BPEL najdemo tudi tretjo različico izvedbe ponavljajočih se segmentov aktivnosti. Podobno kot aktivnosti »While« in »RepeatUntil« tudi aktivnost »ForEach« omogoča izvajanje iteracij, vendar s to razliko, da je v primeru uporabe »ForEach« aktivnosti število iteracij določeno vnaprej. Določanje števila iteracij je doseženo z gnezdenima konstruktoma, ki predstavljata začetno in končno vrednost števca iteracij. Slika 4.15 prikazuje primer konstrukta »ForEach« in definicijo števcev iteracij. Gnezdene aktivnosti so nato izvedene znotraj omejene skupine, ki je predstavljena s konstruktom »Scope«. Ta predstavlja zaprto izvajalno okolje, ki omogoča izolacijo aktivnosti, izvajanih znotraj okolja. Z uporabo aktivnosti »Scope« lahko proces razdelimo na manjše hierarhično organizirane enote. Ponuja podoben koncept, kot ga uporabljamo pri programiranju ob uporabi funkcij. Znotraj funkcije lahko definiramo svoje spremenljivke, ki imajo omejen doseg in živijo samo znotraj omejenega izvajalnega okolja. Konstrukt »Scope« spada v skupno kompleksnih aktivnosti in se lahko uporablja tudi kot samostojen konstrukt, znotraj katerega so gnezdene druge aktivnosti. Posebnost aktivnosti »ForEach« je tudi v tem, da omogoča izvajanje gnezdenih aktivnosti v paralelnem načinu za razliko od njenih sorodnih aktivnosti, pri katerih je omogočeno le zaporedno izvajanje aktivnosti. V primeru uporabe paralelnega načina izvajanja je poleg določanja začetnega in končnega števila iteracij potrebno določiti tudi dodatne podatke, s katerimi definiramo uspešnost izvedbe gnezdenih aktivnosti. Te se v tem primeru izvajajo v obliki vej, kjer imamo

možnost določanja števila izvajalnih vej, ki jih je potrebno uspešno zaključiti, preden lahko nadaljujemo z izvajanjem poslovnega procesa. V primerih, ko se določene veje ne zaključijo uspešno, je možno z uporabo dodatnega parametra nadzorovati samo uspešno izvedene veje.

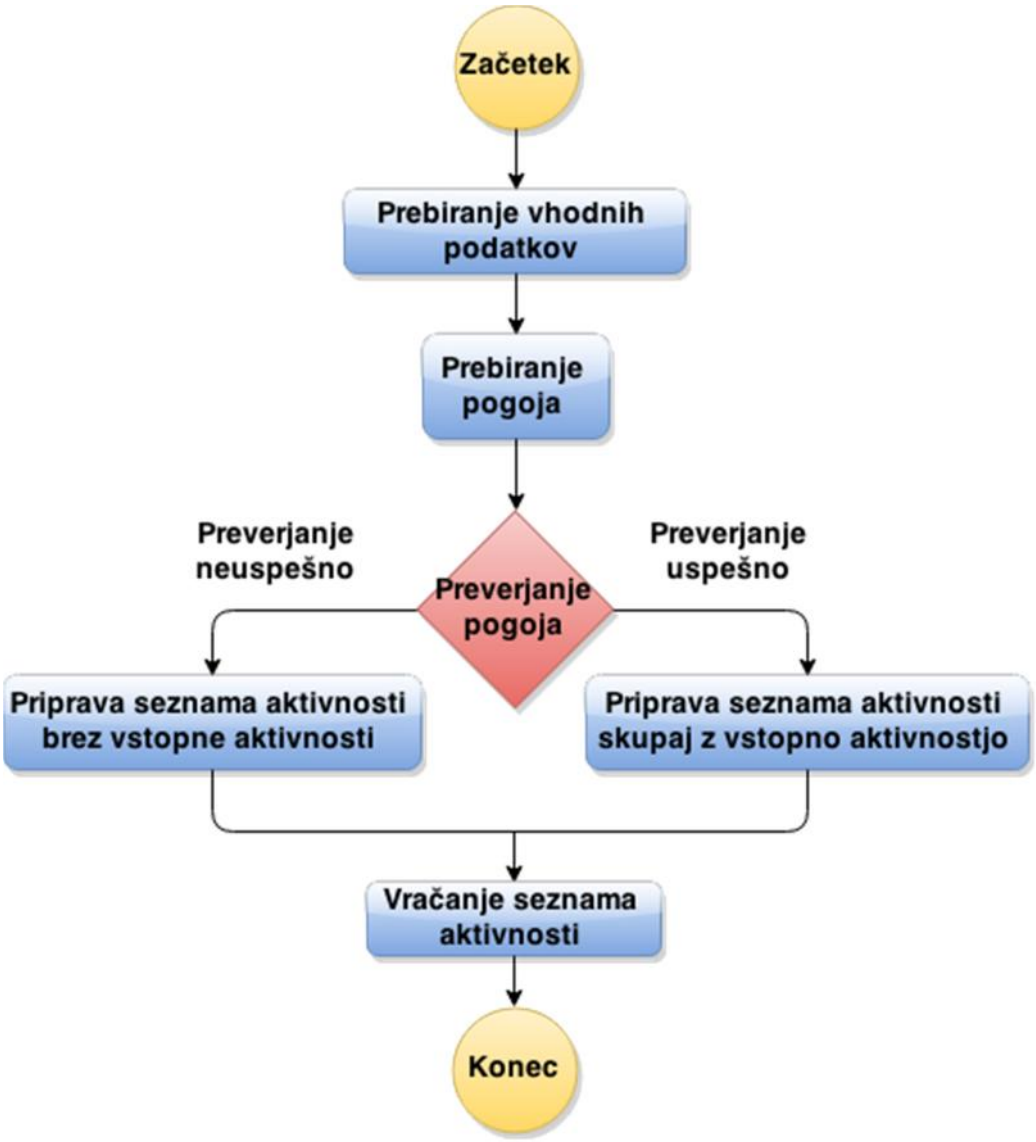



```
<bpel:forEach parallel="no"
      counterName="Counter"
      name="ZankaForEach">
  <bpel:startCounterValue>
      <![CDATA[1]]>
  </bpel:startCounterValue>
  <bpel:finalCounterValue>
      <![CDATA[12]]>
  </bpel:finalCounterValue>
  <bpel:scope>
    <!-- izvedba klica na
    zunanjo spletno storitev
    za podan mesec in izracun
    vsote stroskov porabe
    elekticne energije -->
  </bpel:scope>
</bpel:forEach>
```

Slika 4.15: Aktivnost "ForEach" na primeru izračuna mesečne porabe električne energije, kjer za vsako iteracijo števec "Counter" pridobi vrednost porabe za tekoči mesec (1–12).

4.2.6.2 Preslikava aktivnosti While, RepeatUntil in ForEach

Funkcionalnost zank predstavlja izvajanje pogojno ponavljajočih se segmentov gnezdenih aktivnosti. Tako smo pri preslikavi aktivnosti »While« uporabili že obravnavan model, kjer smo poleg preslikave aktivnosti prilagodili tudi zaporedje izvajanja aktivnosti. Izvajanje posamezne iteracije aktivnosti »While« je pogojeno z veljavnostjo pogoja kot prikazuje slika 4.16. Ob uspešni evalvaciji pogoja se izvede zaporedje gnezdenih aktivnosti. Izvajanje iteracij se prekine, ko pogoj ni več veljaven.



Slika 4.16: Diagram poteka izvedbe aktivnosti "While" s preverjanjem veljavnosti pogoja.

Aktivnost »While« in njene iteracije smo podprli s konceptom rekurzivnega izvajanja (Slika 4.17). Začetek izvajanja aktivnosti »While« je pogojen z veljavnostjo pogoja. V primeru veljavnega pogoja smo pridobili seznam gnezdenih aktivnosti in nadaljevali z izvajanjem slednjega. Pridobljene podatke smo nato uporabili za vsako iteracijo rekurzije. Da lahko preverjamo veljavnost pogoja, smo poleg gnezdenih aktivnosti kot vhodni parameter med iteracijami posredovali tudi samo aktivnost »While«. V primeru, da pogoj ni izpolnjen, smo začeli z rekurzivnim vračanjem rezultatov do zadnjega nivoja. Na tak način smo kot rezultat aktivnosti »While« pridobili rezultat rekurzivnega izvajanja gnezdenih aktivnosti. Po zaključku rekurzije smo z obdelavo aktivnosti »While« zaključili in lahko nemoteno nadaljevali z izvajanjem naslednje aktivnosti glede na vrstni red.

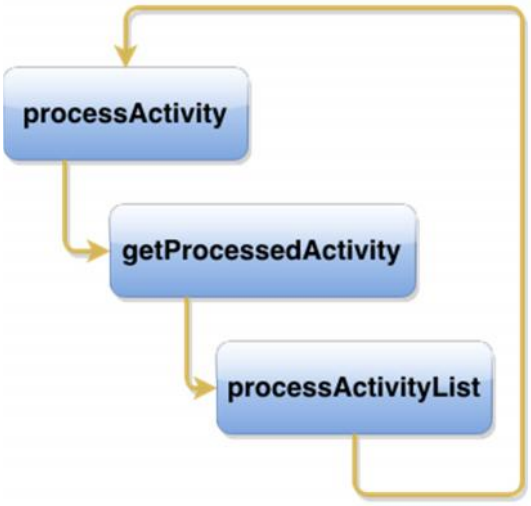
```
case WHILE:
      Promise<List<TActivity>> whileActivity = operations.processWhile(activity,
                                                        variablesHolder,
                                                        variableValuesHolder);
      return this.getProcessedActivity(whileActivity, operations.getVariables(),
                                       operations.getVariableValues(),
                                       operations.getAdditionalOptions());

case FOREACH:
      Promise<List<TActivity>> scopeActivityList =
      operations.processForEach(activity, variablesHolder, variableValuesHolder);
      return this.getProcessedActivity(scopeActivityList,
                                       operations.getVariables(),
                                       operations.getVariableValues(),
                                       operations.getAdditionalOptions());

case REPEATUNTIL:
      Promise<List<TActivity>> repeateUntilActivityList =
      operations.processRepeatUntilCondition(activity, variablesHolder,
                                             variableValuesHolder);
      return this.getProcessedActivity(repeateUntilActivityList, variablesHolder,
                                       variableValuesHolder,
                                       operations.getAdditionalOptions());
```

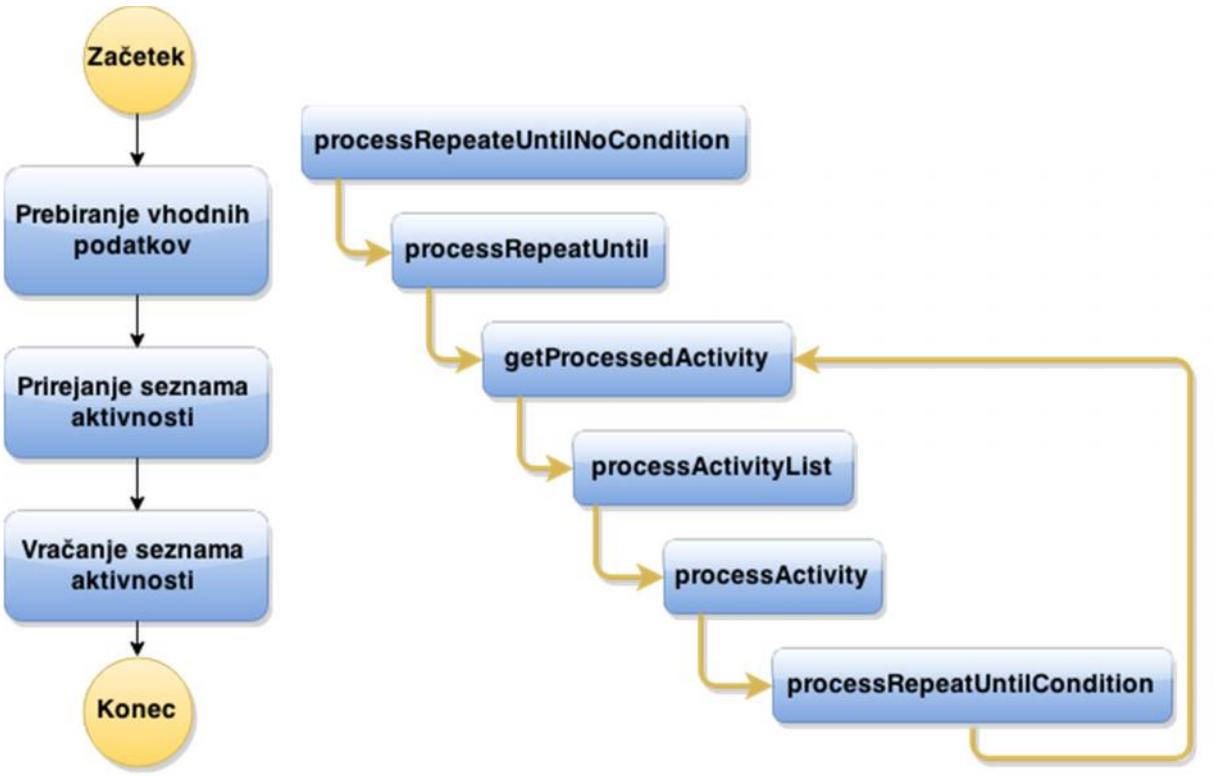
Slika 4.17: Preslikava aktivnosti »While«, »ForEach« in »RepeatUntil« v obliki klicev funkcij, ki vsebujejo implementacijo posamezne aktivnosti.

Poleg rekurzivnega obstaja tudi pristop, kjer za potrebe gnezdenja izdelamo tako imenovani gnezdeni delovni tok – podtok. V tem primeru se izvajanje iteracij preseli v drug delovni tok, kjer se izvedejo tudi vse gnezdene aktivnosti. Glavni tok je pri tem (podobno kot v primeru rekurzivnega pristopa) čakal na končanje podtoka. Po končani izvedbi in pridobljenih rezultatih podtoka lahko nadaljujemo z izvedbo glavnega toka. V primeru uporabe zank smo vedno uporabljali koncept rekurzije, kot je prikazan na sliki 4.18.



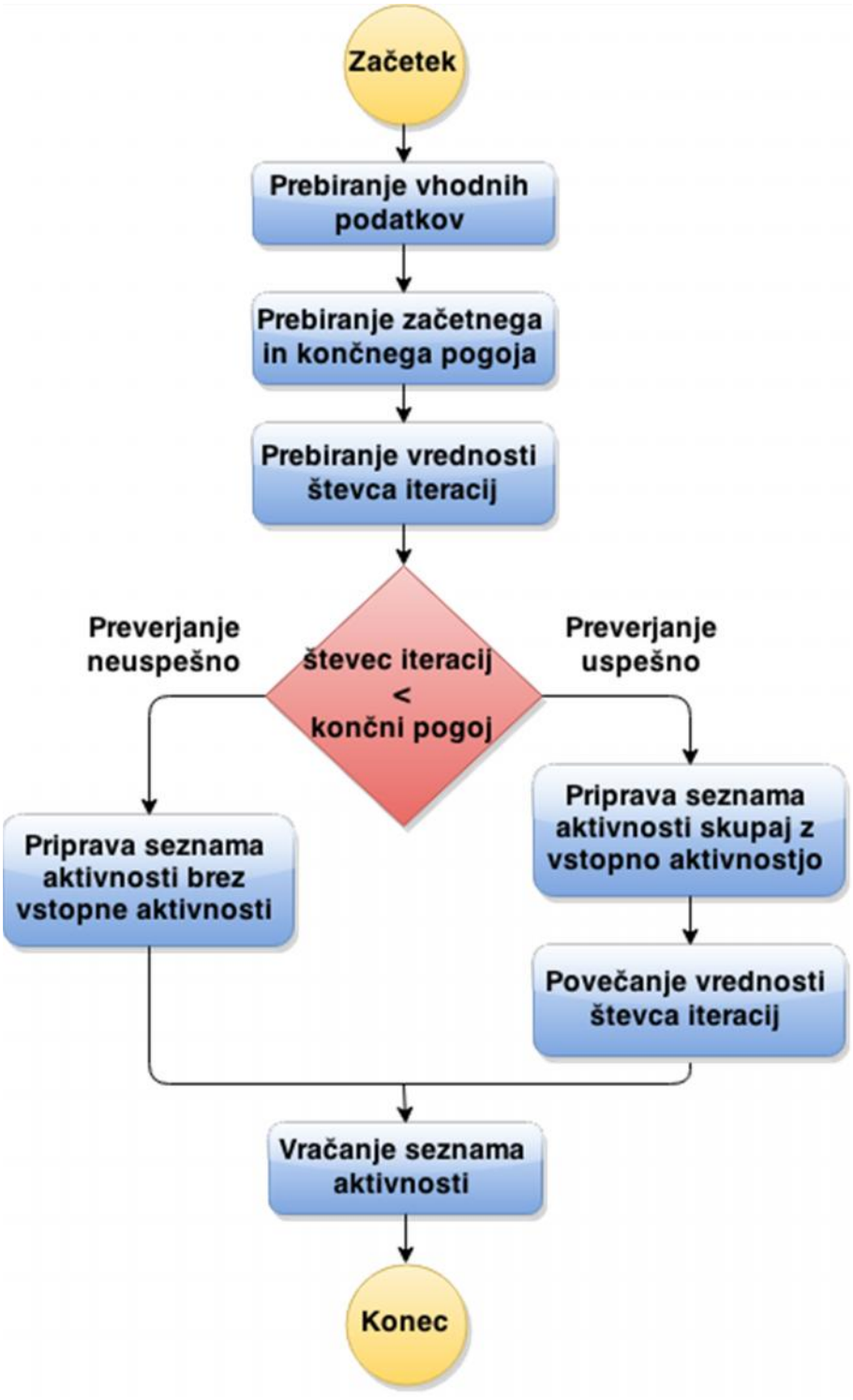
Slika 4.18: Pristop rekurzivnega izvajanja seznama aktivnosti v primeru uporabe zank.

Na podoben način kot aktivnost »While« smo preslikali tudi aktivnost »RepeatUntil« (Slika 4.17). Omenjeni aktivnosti se med seboj razlikujeta predvsem po položaju pogoja, saj se pogoj pri aktivnosti »While« nahaja na prvem mestu – pred izvedbo gnezdenih aktivnosti, medtem ko se pri aktivnosti »RepeatUntil« pogoj nahaja za izvedbo gnezdenih aktivnosti. Druga posebnost aktivnosti »RepeatUntil« je uporaba kompleksne aktivnosti »Sequence«, ki določa zaporedje izvajanja gnezdenih aktivnosti. V tem primeru je aktivnost »Sequence« obravnavana kot kompleksna aktivnost, ki spremeni vrstni red izvajanja aktivnosti. Preslikavo aktivnosti smo zaradi preverjanja pogoja po izvedbi gnezdenih aktivnosti razdelili na dve ločeni funkciji. Prva funkcija – »processRepeateUntilNoCondition« ima nalogo izluščiti seznam gnezdenih aktivnosti (Slika 4.19). Na podlagi seznama izluščenih aktivnosti smo ob izvedbi vsake iteracije gnezdenih aktivnosti izvedli tudi drugo funkcijo – »processRepeateUntilCondition«. Ta prav tako poskrbi za luščenje gnezdenih aktivnosti in poleg tega še preverjanje veljavnosti pogoja. Če je bil pogoj neveljaven, smo prenehali z izvajanjem gnezdenih aktivnosti in po strukturi rekurzivnih klicev vrnili rezultat, ki je veljal kot končni rezultat aktivnosti »RepeatUntil«. V nasprotnem primeru smo nadaljevali z izvajanjem iteracij. Vrstni red izvajanja aktivnosti poslovnega procesa se je podobno kot pri aktivnosti »While« spremenil zaradi gnezdenih aktivnosti in njihovih iteracij.



Slika 4.19: Diagram poteka izvedbe aktivnosti »RepeatUntil« brez preverjanja pogoja (levo) in zaporedje rekurzivnih klicev ob izvajanju omenjene aktivnosti (desno).

Aktivnost »ForEach« predstavlja še eno različico zank, ki pa se od prehodnih dveh razlikuje po številu iteracij. Pri aktivnostih »While« in »RepeatUntil« je število iteracij nedefinirano in se lahko spremeni na podlagi gnezdenih elementov. Pri aktivnosti »ForEach« je število iteracij določeno z začetnim in končnim števcem. Preslikavo aktivnost v okviru delovnih tokov spletne storitve SWF smo izvedli na podoben način kot pri drugih dveh zankah, kot je razvidno s slike 4.20.



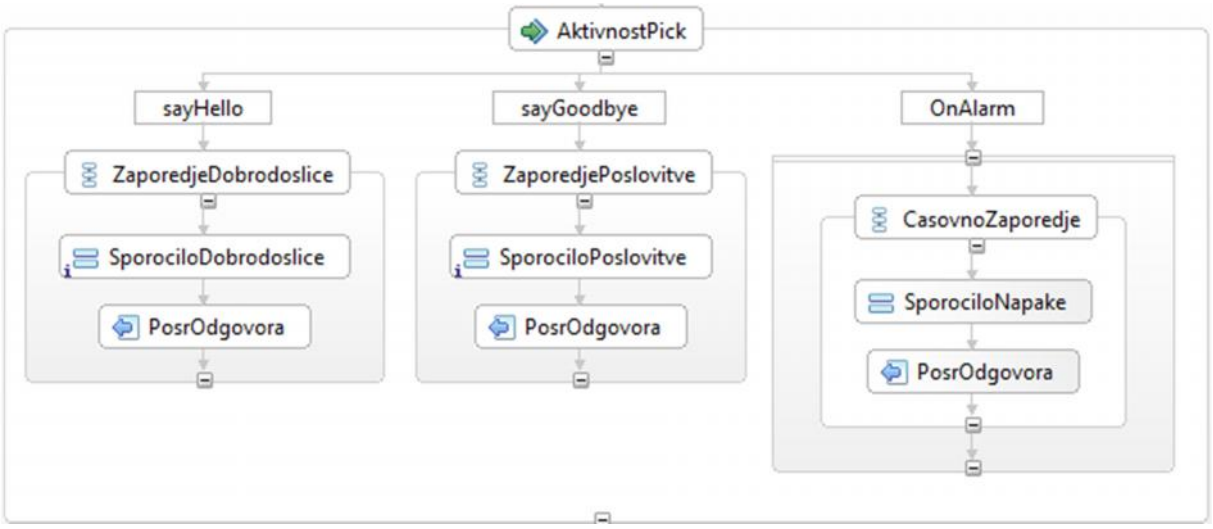
Slika 4.20: Diagram poteka izvedbe aktivnosti "ForEach".

Pred vsako iteracijo smo opravili identifikacijo in preverjanje stanje števca iteracij ter končnega pogoja. Če sta se vrednosti ujemali – števec iteracij je manjši od končnega pogoja, se je izvajanje gnezdenih aktivnosti nadaljevalo v obliki iteracije. Slednje smo podobno kot pri aktivnosti »While« izvedli v obliki rekurzivnih klicev, kjer smo pred začetkom vsake iteracije preverili veljavnost pogoja. Če smo ugotovili, da je pogoj neveljaven, smo izvajanje aktivnosti »ForEach« prekinili, vrednosti gnezdenih spremenljivk pa priredili vrednosti same aktivnosti. Zaradi gnezdenja se je spremenilo zaporedje izvajanja aktivnosti na nivoju poslovnega procesa. Z uporabo rekurzivnega pristopa smo regulirali sam vrstni red izvajanja aktivnosti (Slika 4.17). Posebnost aktivnosti »ForEach« je tudi v možnosti izvedbe gnezdenih aktivnosti v paralelnem načinu. Pri paralelnem pristopu namesto zaporednega izvajanja aktivnosti v rekurzivnih iteracijah le-te prožimo paralelno. Število paralelno sproženih aktivnosti je odvisno od končnega pogoja. Rezultat paralelno sproženih aktivnostih sprejemamo v asinhronem načinu, kjer je potrebno počakati na rezultat izvedbe posameznih aktivnosti. Na podlagi dodatnih parametrov lahko določimo potrebno število izvedenih gnezdenih aktivnosti in omejimo čakanje na odgovore asinhronih klicev pri izvedbi enega cikla.

4.2.7 Aktivnost Pick

4.2.7.1 Opis aktivnosti Pick

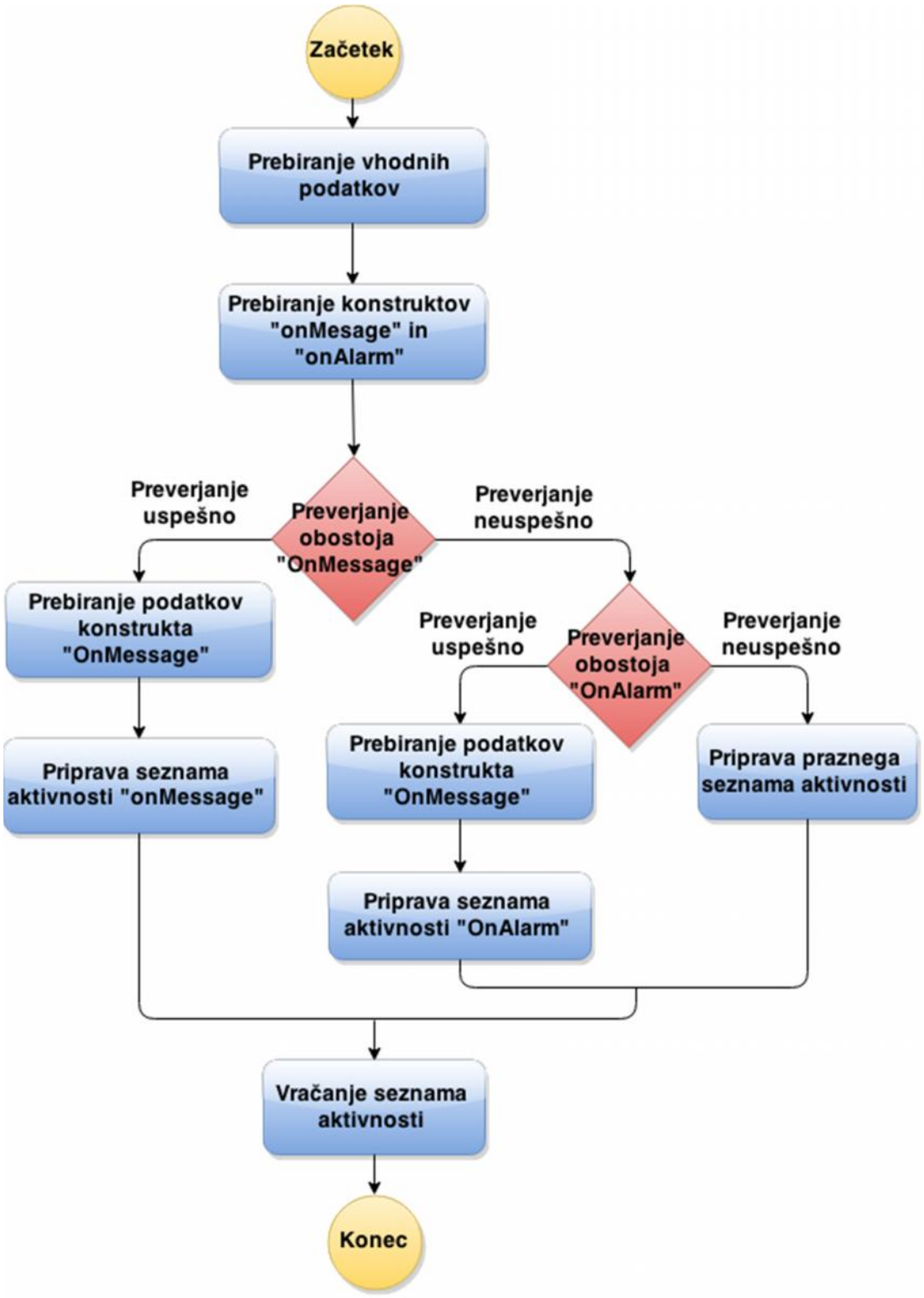
Aktivnost, s katero lahko definiramo pogoje proženja poslovnega procesa, je definirana s konstruktom »Pick«. Omenjen konstrukt lahko zamenja funkcionalnost aktivnosti »Receive«, saj omogoča proženje nove instance poslovnega procesa. Ta aktivnost omogoča zagon poslovnega procesa na podlagi sporočilnih ali časovnih dogodkov. Sporočilni dogodek je definiran s konstruktom »onMessage« in je (po funkcionalnosti) enak konstruktu »Receive«, obenem pa prav tako vsebuje vse njegove lastnosti (tip vrat in operacije spletne storitve ter spremenljivko za prihajajoče sporočilo). Omogoča proženje poslovnega procesa na podlagi sporočila, prispelega preko definirane operacije. Časovni dogodek je definiran s konstruktom »OnAlarm« in omogoča proženje poslovnega procesa glede na čas prispetja sporočila. Proženje je časovno omejeno na pretek določenega časovnega obdobja ali na proženje ob točno določenem času. Za določitve se uporabljata časovna formata »dateTime« in »date«. Slika 4.21 predstavlja primer aktivnosti »Pick« z uporabo konstruktov »onMessage« in »onAlarm«.



Slika 4.21: Aktivnosti "Pick", kjer se na podlagi konstrukta "onMessage" uporabniku zaželi sporočilo dobrodošlice ali poslovitve. V primeru zakasnitve se posreduje odgovor z napako.

4.2.7.2 Preslikava aktivnosti Pick

Pri izvedbi preslikave aktivnosti »Pick« (Slika 4.22) smo uporabili podoben pristop kot pri preslikavi aktivnosti »Receive«. Glede na to, da gre za relativno podobno funkcionalnost pri obeh aktivnostih, smo uporabili podobno implementacijo pri preverjanju tipa vrat in operacije. Bistveno razliko predstavljajo dodatni konstrukti aktivnosti »Pick«, ki so vezani na sporočila oz. čas. Na nek način gre za princip »kdor prej pride, prej melje«, v smislu izvedbe konstrukta, ki je definiran z »onMessage« ali konstrukta »onAlarm«.



Slika 4.22: Diagram poteka izvedbe aktivnosti »Pick« s preverjanjem konstruktov "onMessage" in "onAlarm".

Vsak izmed omenjenih konstruktov vsebuje gnezdene aktivnosti v obliki zaporedja, definiranega z aktivnostjo »Sequence«. Aktivnost »Pick« deluje kot poslušalec na dogodek – sporočilo oz. deluje na časovni bazi. Na podlagi dogodka, ki se časovno gledano zgodi prvi,

se izvede eden izmed prej omenjenih pripadajočih konstruktov. Konstrukt »OnMessage« je podobno kot aktivnost »Receive« vezan na zunanjo storitev, zato se tudi njuna preslikava ujema. Drugačna preslikava velja za konstrukt »onAlarm«, ki je sprožen ob preteku določenega časa oz. ob določenem času. Merjenje časa je povezano s sistemsko uro, kjer se implementacija nahaja. Oba konstrukta vsebujeta gnezdene aktivnosti. Ob proženju določenega konstrukta se v rekurzivnem načinu izvedejo vse aktivnosti, definirane v konstruktu »Sequence«. Postopek izvedbe je podoben tistemu pri aktivnostih »While«, »RepeatUntil« in »ForEach«, kar je razvidno tudi s slike 4.23. Ob zaključku izvedbe gnezdene aktivnosti se rezultati prenesejo na aktivnosti »Pick«.

```
case PICK:

      Promise<PickObject> customActivity = operations.processPick(activity,
                                                          variablesHolder,
                                                          variableValuesHolder,
                                                          additionalOptions);

      sequenceActivityList = processCustomActivity(customActivity);
      return this.getProcessedActivity(sequenceActivityList,
                                       operations.getVariables(),
                                       operations.getVariableValues(),
                                       operations.getAdditionalOptions());
```

Slika 4.23: Preslikava aktivnosti »Pick« v obliki klica funkcije, ki vsebujejo implementacijo aktivnosti.

5 Primer realizacije pretvorbe na podlagi modela

Koncept preslikave poslovnih procesov jezika BPEL na storitve, ki se izvajajo na računalniškem oblaku, smo potrdili z izdelavo prototipne aplikacije. Izdelana aplikacija služi kot primer in potrditev koncepta, s pomočjo katerega preslikamo obstoječe procese jezika BPEL na delovne tokove spletne storitve SWF podjetja Amazon. Poleg same preslikave poslovnega procesa na delovne tokove je podprto tudi izvajanje poslovnega procesa. Postopek poteka izvedb posameznih aktivnosti in njihovih gnezdenih aktivnosti je dosegljiv preko nadzorne plošče spletne storitve SWF.

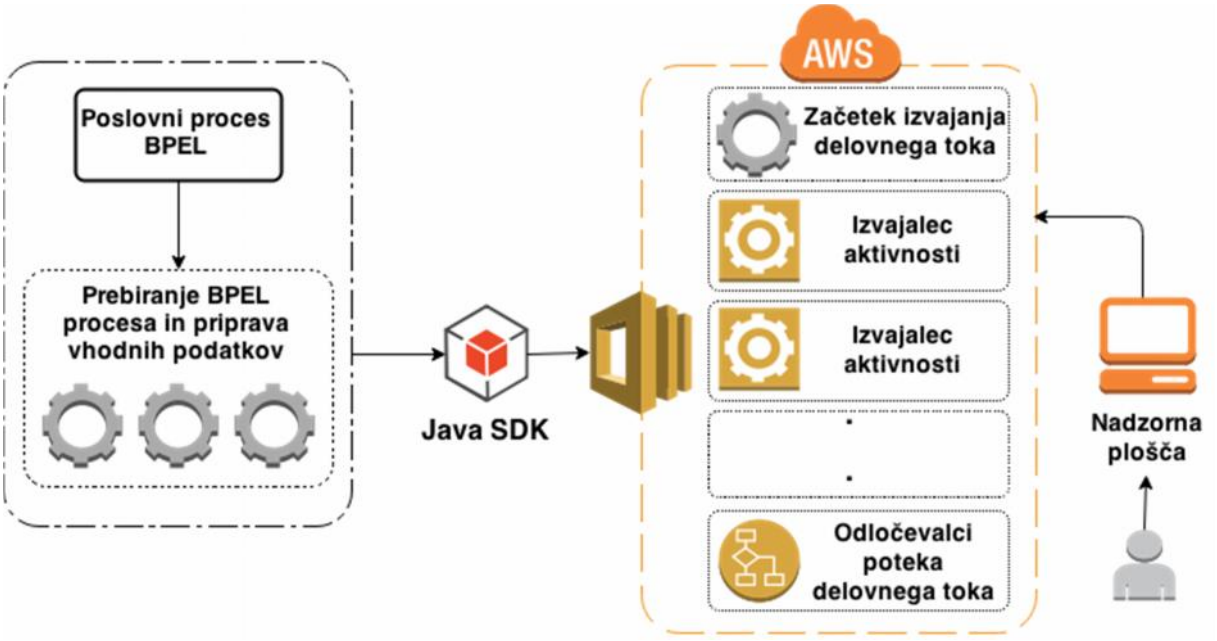
Izdelavo aplikacije za potrditev koncepta smo začeli pri poslovnih procesih jezika BPEL. Za izvedbo analize in kasneje tudi same preslikave smo potrebovali primere procesov jezika BPEL. V ta namen smo uporabili razvojno orodje Eclipse, s pomočjo katerega smo kreirali večje število testnih procesov, pri tem so štirje podrobneje obravnavani v nadaljevanju (poglavje 6.2). Nabor testnih procesov so sestavljali: (1) procesi, ki so vsebovali posamezne aktivnosti (procesi v obliki samostojnih aktivnosti »Assign«, »If«, »While«), in (2) kompleksnejši procesi, ki so sestavljeni iz večjega števila aktivnosti (proces spletne menjalnice, porabe električne energije …). Omenjeno orodje smo razširili z uporabo raznih dodatkov, ki so omogočali kreiranje in kasneje tudi izvajanje poslovnih procesov. V našem primeru smo orodje razširili z dodatkom Eclipse BPEL Designer, ki omogoča kreiranje in s pomočjo dodatnih nastavitev tudi izvajanje poslovnih procesov. Za slednje je potrebna dodatna nadgradnja v vidu procesnega strežnika, ki podpira izvajanje poslovnih procesov. Za namestitev omenjenega strežnika smo uporabili programsko rešitev Apache ODE, ki skrbi za življenjsko okolje in izvajanje poslovnih procesov, sprejemanje in pošiljanje sporočil s strani spletnih storitev itd. Izvajanje poslovnih procesov na lokalnem razvojnem okolju smo uporabili v procesu verifikacije. V sklopu verifikacije smo opravili primerjavo med procesi, izvedenimi v razvojnem okolju, in procesi, ki so izvedeni kot delovni tokovi spletne storitve SWF. Uporabljene aktivnosti v testnih primerih smo analizirali glede na izdelane procese jezika BPEL, kjer smo določene aktivnosti uporabili tudi večkrat in s tem poskušali pokriti vse variacije funkcionalnosti.

Spletna storitev SWF, ki je dostopna kot ena izmed storitev računalniškega oblaka podjetja Amazon, ima za komunikacijo omogočene različne vmesnike, implementirane v več različnih programskih jezikih. Prav tako je s strani podjetja Amazon za uporabo njihovih storitev na voljo več paketov za razvoj programske opreme (*SDK – Software Development Kit*) v različnih programskih jezikih. Paket programske opreme s podporo programskega jezika Java [30] velja za enega najbolj priljubljenih, kar je obenem tudi glavni razlog za njegov izbor. Gre za objektno usmerjen programski jezik, ki temelji na razredih. Velika prednost programskega jezika Java je zmožnost, da se programi, napisani v omenjenem jeziku, lahko izvajajo povsod,

neodvisno od arhitekture računalnika. Da je takšno izvajanje sploh možno, gre velika zahvala javanskem navideznemu stroju (*JVM – Java Virtual Machine*). Ta služi kot vmesna plast, kjer se javanska koda prevede v bitno kodo, ki se izvaja na okolju JVM, kar predstavlja idealno rešitev tako za uporabnika kot tudi za razvijalca. Slednji mora poskrbeti za izvedbo in nemoteno delovanje znotraj enega samega okolja – JVM. Izvajanje na vseh ostalih stacionarnih in prenosnih napravah je z uporabo JVM avtomatsko zagotovljeno.

5.1 Arhitekturna zasnova rešitve

Na podlagi izbire programskega jezika in idejne zasnove, prikazane na sliki 5.1, smo izdelali načrt, ki zajema razvoj funkcionalnosti po sklopih. Sklope smo glede na vsebino razdelili v več skupin.



Slika 5.1: Idejna zasnova koncepta preslikave procesa jezika BPEL na delovne tokove spletne storitve SWF.

Tako so nastale tri glavne skupine: (1) obdelava procesa jezika BPEL, (2) izvajanje procesa v obliki delovnih tokov spletne storitve SWF in (3) skupne funkcionalnosti. Vsebino skupine so tvorili posamezni projekti, predstavljeni kot moduli, ki so pokrivali določen del funkcionalnosti. Prva skupina funkcionalnosti je izdelana v obliki modulov, ki predstavljajo ogrodje za nadaljnje operacije, kjer smo podprli funkcionalnost dela z raznimi datotekami tipa WSDL, XML, XSD, na podlagi katerih smo pridobili različne (vhodne) podatke. V to skupino modulov spadajo:

- »BPELActivityObject« – kreiranje javanskih objektov na podlagi predloge XSD definicije jezika BPEL,
- »BPELObjectMapper« – preslikava med notacijo XML in javanskimi objekti,

- »BPEL_WS« – kreiranje javanskih objektov na podlagi pripadajoče spletne storitve.

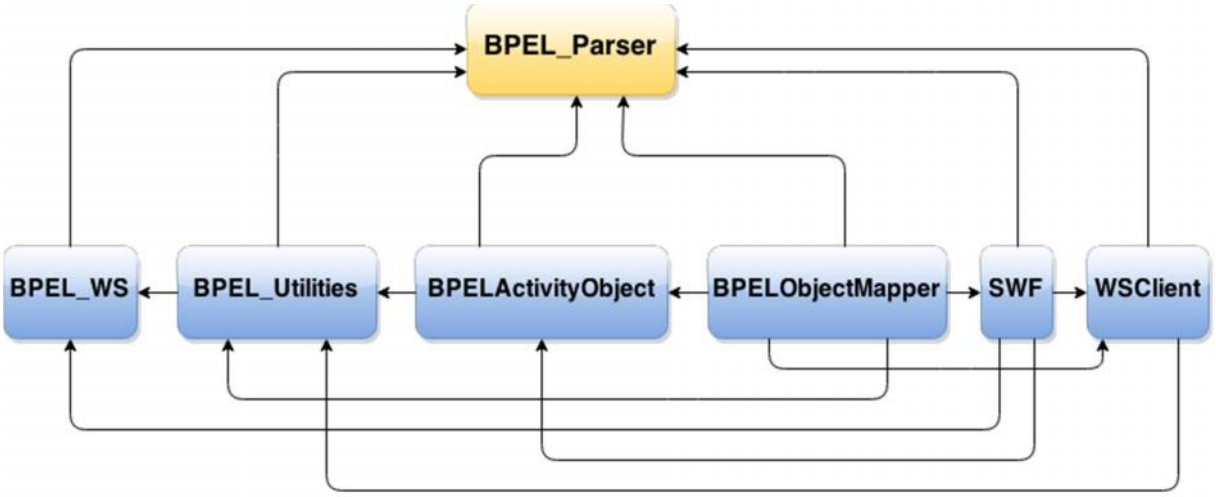
Sledila je druga skupina modulov, ki vsebuje implementacijo posameznih aktivnosti, pridobljenih s pomočjo modulov iz predhodne skupine. Implementacija posameznih aktivnosti je izdelana po načelih in pravilih, definiranih s strani spletne storitve SWF:

- »SWF« – implementacija konstruktov delovnih tokov spletne storitve SWF, ki predstavljajo posamezne aktivnosti jezika BPEL; izvajanje delovnih tokov,
- »WSClient« – generičen odjemalec spletne storitve.

Znotraj teh modulov je bila implementirana vsa potrebna funkcionalnost za uspešno izvajanje procesov jezika BPEL v obliki delovnih tokov spletne storitve SWF. Zadnja skupina je prestavljala knjižnico funkcionalnosti, katere nabor je uporabljen pri več ostalih modulih (modul »BPEL_Utilities«). Ob združitvi vseh treh sklopov smo pridobili programsko ogrodje, ki je na podlagi vhodnih podatkov skrbelo za pretvorbo poslovnega procesa in pravilno izvajanje v obliki delovnih tokov spletne storitve SWF.

Za lažje medsebojno povezovanje in upravljanje med seboj ločenih funkcionalnosti smo se odločili uporabiti že omenjen koncept izdelave modulov, pri čemer več modulov skupaj predstavlja celotno rešitev. Pri implementaciji slednjih smo uporabili koncept modulov Maven [22]. Celotna rešitev tako temelji na projektu Maven, ki vsebuje množico manjših modulov. Glavni projekt je predstavljen kot nadrejen projekt vsem ostalim modulom, ki so med seboj povezani, kot prikazuje slika 5.2. Struktura projekta in modulov je tipično hierarhična, kjer projekt predstavlja glavni člen – očeta, moduli pa so predstavljeni kot podčleni – sinovi. S tem pristopom dosežemo lažje upravljanje modulov in enostavnejše deljenje lastnosti, ki so skupne vsem modulom. Z uporabo koncepta modulov Maven prebrodimo številne težave, ki nastanejo med postopkom implementacije. Ena izmed glavnih prednosti uporabe koncepta modulov Maven je upravljanje odvisnosti med projekti in knjižnicami. V našem primeru se je to izkazalo za zelo pomemben faktor, saj se določene funkcionalnosti in knjižnice lahko na enostaven način delijo med različne module. Z uporabo deljenih odvisnosti poskrbimo, da določeno implementacijo lahko uporabljamo pri več različnih modulih, ne da bi naleteli na problem podvajanja funkcionalnosti. S takim pristopom se obenem izognemo tudi ponavljajočim se napakam, saj implementacijo določene funkcionalnosti definiramo na enem mestu, uporabimo pa večkrat, glede na potrebe. Uporaba modulov Maven pa s seboj prinaša tudi druge prednosti. Ena takih je tudi izboljšana sistematizacija in enostaven pristop k avtomatizaciji postopkov sistema za gradnjo programja. Pri tem se uporabljajo nastavitvene datoteke tako imenovane datoteke POM (*Project Object Module*) [23]. Te so zapisane v obliki notacije XML. Namenjene so konfiguriranju posameznih nastavitev in medsebojnih odvisnosti na nivoju posameznega modula. Uporabljajo se v različnih fazah življenjskega cikla določenega modula. Z uporabo

nastavitvenih datotek lahko na enostaven način sprožimo postopek kreiranja javanskih objektov na podlagi predloge XSD, kreiramo spletno storitev, definirano v datoteki WSDL in podobno. Dejansko gre za približek postopka avtomatizacije z uporabo določenih nastavitev. Ta pristop smo uporabljali na različnih nivojih za reševanje različnih težav, povezanih s kreiranjem osnovnih gradnikov ali pa z dejanskim izvajanjem aplikacije.



Slika 5.2: Hierarhija modulov Maven, kjer puščice predstavljajo smer dedovanja odvisnosti.

5.2 Obdelava poslovnega procesa jezika BPEL

Postopek pretvorbe poslovnih procesov predstavlja zapleten proces, ki je zastavljen zelo široko in dinamično. Namenjen je uporabnikom z bolj naprednim poznavanjem poslovnih procesov in postopkov izvajanja. Izhodišče predstavlja poslovni proces BPEL, ki služi kot podatek, ki ga je treba pravilno obdelati in prilagoditi, da bo lahko uporaben za potrebe izvajanja v drugačnem okolju.

Za pravilno izvedbo preslikave poslovnega procesa je treba upoštevati določena navodila, ki so v našem primeru definirana v obliki predloge XSD. V ta namen smo izdelali modul »BPELActivityObject«, ki služi kreiranju javanskih razredov na podlagi predloge XSD. Generirani javanski razredi predstavljajo posamezne aktivnosti in njihove povezave. Na podlagi teh aktivnosti smo za potrebe preslikovanja posameznega poslovnega procesa izdelali ločen modul – »BPELObjectMapper«. Vsakemu poslovnemu procesu pripada tudi spletna storitev – odjemalec, ki sproži začetek izvajanja poslovnega procesa. Za potrebe pridobivanja podatkov omenjene spletne storitve smo izdelali implementacijo v obliki modula – »BPEL_WS«.

5.2.1 Preslikava definicije poslovnih procesov

Preslikavo definicije posameznih aktivnosti poslovnih procesov, zapisanih v obliki predloge XSD v javanske objekte, smo podprli z uporabo knjižnice JAXB (*Java Architecture for XML Binding*) [18]. Ta na podlagi podane predloge XSD (definicija jezika BPEL, standardizirana s

strani organizacije OASIS) avtomatsko kreira ustrezne javanske razrede. Slednji vsebujejo vse atribute, potrebne za neposredno preslikavo oz. normalizacijo dokumenta v obliki notacije XML v javanski razred. V praksi to pomeni, da bodo vsi elementi, atributi in lastnosti neke značke XML preslikani v javanski razred. Ta bo vseboval potrebne spremenljivke in dinamične strukture, s katerimi bo podprta funkcionalnost, ki je predstavljena z določeno značko XML. Na ta način je zagotovljena podpora variacije določene aktivnosti in hranjenje podatkov v pravilnem formatu. Rezultat preslikave v javanski objekt je prikazan na sliki 5.3 na primeru aktivnosti »Assign«. V določenih primerih preslikava s pomočjo knjižnice JAXB ne zadostuje, saj preslikava določenih lastnosti posamezne aktivnosti ni veljavna (npr. preslikava imenskega prostora v ime paketa javanskega objekta). Za veljavno izvedbo preslikave je potrebna razširitev v okviru dodatnih nastavitev preko tako imenovanih vezav JAXB (*JAXB bindings*) [11]. Tovrstnih dodatkov smo se posluževali v primerih, kjer smo morali zagotoviti pravilno kreiranje imenskih prostorov paketov javanskih razredov na podlagi notacij XML. Z uporabo omenjene knjižnice JAXB in njenih razširitev v obliki vezav smo dosegli preslikavo vseh aktivnosti v med seboj neodvisne javanske razrede, ki so uporabljeni v kasnejših fazah razvoja. Poleg avtomatskih preslikav smo za potrebe določene funkcionalnosti, ki so skupne več razredom, kreirali poseben javanski razred, s katerim smo razširili funkcionalnost vseh ostalih razredov. Možnost razširitve funkcionalnosti smo uporabili pri implementaciji funkcionalnosti hranjenja vrednosti spremenljivk. V ta namen smo izdelali dve dinamični strukturi za hranjenje vrednosti spremenljivk, kjer smo eno uporabili za hranjenje imena in tipov spremenljivk ter drugo za hranjenje imena in vrednosti spremenljivk. Omenjeni dinamični strukturi sta bili predstavljeni kot javanski slovar – »Java Map« s ključi in pripadajočimi vrednostmi. Ključi omenjenih dinamičnih struktur so vsebovali imena spremenljivk, vrednost pa imena tipov spremenljivk oz. njihovih vrednosti. Funkcionalnost hranjenja spremenljivk, njihovih tipov in vrednosti je bila uporabljena v kasnejših fazah za hranjenje spremenljivk jezika BPEL.

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "tAssign", propOrder = {
    "copyOrExtensionAssignOperation"
})
public class TAssign
    extends TActivity
    implements Serializable, ToString {

    private final static long serialVersionUID = 1L;
    @XmlElements({
        @XmlElement(name = "copy", type = TCopy.class),
        @XmlElement(name = "extensionAssignOperation", type =
TExtensionAssignOperation.class)
    })
    protected List<TExtensibleElements> copyOrExtensionAssignOperation;
    @XmlAttribute(name = "validate")
    protected TBoolean validate;
```

Slika 5.3: Primer generiranega javanskega razreda aktivnosti »Assign« na podlagi notacije XML.

5.2.2 Preslikava poslovnega procesa v javanske razrede

Za potrebe preslikave obstoječih poslovnih procesov jezika BPEL na delovne tokove spletne storitve SWF smo implementirali rešitev, ki poskrbi za prebiranje dokumenta XML (ki vsebuje strukturo procesa jezika BPEL) in na podlagi vsebine kreira javanske objekte. Ti objekti hranijo pravilno strukturo in vse lastnosti posamezne aktivnosti poslovnega procesa, pa tudi vse njene variacije.

Na podlagi konfiguracije v obliki nastavitvene datoteke uporabnik poda pot do poslovnega procesa, zapisanega z jezikom BPEL, in pripadajoče definicije spletne storitve – WSDL. Za oba tipa datotek je vnaprej določeno privzeto mesto, kjer naj bi se nahajali omenjeni datoteki. Na podlagi podanih vhodnih podatkov sledi postopek prebiranja datotek. Prebiranje omenjenih datotek poteka na različne načine glede na tip datoteke. Datoteka, ki vsebuje poslovni proces jezika BPEL, predstavlja vsebino poslovnega procesa, ki se bo izvedel. Nabor različnih aktivnosti, ki si sledijo v nekem logičnem zaporedju, predstavlja poslovno logiko procesa. Prebiranje poslovnega procesa je izvedeno s pomočjo knjižnice JAXB, kot je razvidno s slike 5.4. Ob prebiranju vsebine poslovnega procesa smo določene aktivnosti preslikali neposredno, saj se definicije aktivnosti v notaciji XML povsem prilagajajo nastalim lastnostim objektov v programskem jeziku Java. A povsod vendarle ni tako. Določene lastnosti so na strani javanskih objektov zapisane v različnih dinamičnih strukturah, da bi podprle zahtevano funkcionalnost. Težave, ki so pri tem nastale, so se pojavile kasneje, pri implementaciji v povezavi z delovnimi tokovi spletne storitve SWF, in so opisane v nadaljnjih poglavjih. Nekatere izmed aktivnosti so vsebovale tudi podatke, vezane na pripadajoče spletne storitve in druge zunanje povezave. Za takšne primere aktivnosti smo implementirali dodatne funkcionalnosti preverjanja tako na nivoju same preslikave med notacijo XML in

javanskimi objekti kot tudi na vsebinskem nivoju. Pri slednjem je to predstavljeno kot dodatno preverjanje in uparjanje poslovnega procesa z zunanjimi spletnimi storitvami. Povezovanje z njimi je obenem lahko tudi del funkcionalnosti določene aktivnosti – »Receive«, »Invoke«, »Reply« in drugih. Pri uparjanju aktivnosti, ki vsebujejo povezave z zunanjimi spletnimi storitvami, smo se osredotočili predvsem na konstrukt »PartnerLink« (predstavlja glavni člen za povezovanje) in njegove lastnosti. Pri tem smo opravljali uparjanje po različnih atributih – »PartnerLink«, »portType«, »operation«. Določeni od naštetih atributov so definirani tako na strani poslovnega procesa (kot lastnosti aktivnosti) kot tudi spletne storitve. Uparjanje je uspešno, če se vse vrednosti omenjenih atributov na obeh straneh ujemajo. V tem primeru ima določena aktivnost veljaven dostop do zunanje spletne storitve.

```
public XMLObjectMapper()
{
	super();
	this.bpelFileName =
		this.getAppConfigProperties().getProperty("bpelFilePath");
	this.bpelFolderName =
		this.getAppConfigProperties().getProperty("bpelFolder");

	ClassLoader classloader = Thread.currentThread().getContextClassLoader();
	this.bpelFileInputStream =
				classloader.getResourceAsStream(this.bpelFileName);
	this.mapBPELtoObjects();
}

private void mapBPELtoObjects(InputStream bpelInputStream)
{
	JAXBContext jaxbContext;
	try
	{
		//	1. Kreiranje JAXBContenxt-a na podlagi imena paketa
		jaxbContext = JAXBContext.newInstance("bpel.xsd.types.generated");
		//	2. Uporaba instance JAXBContext-a za kreiranje Unmarshaller-ja
		Unmarshaller unmarshaller = jaxbContext.createUnmarshaller();
		//	3. Uporaba Unmarshaller-ja za prebiranje dokumenta XML
		Object unmarshalled = unmarshaller.unmarshal(bpelInputStream);
		TProcess processObj = (TProcess)
					JAXBIntrospector.getValue(unmarshalled);
		//	4. Prirejanje pridobljene instance procesa
		this.bpelProcess = processObj;
	} catch (JAXBException e) {
		log.error(e.getMessage());
	}
}
```

Slika 5.4: Prebiranje datoteke BPEL in preslikovanje aktivnosti v javanske objekte s pomočjo knjižnice JAXB.

5.2.3 Preslikava pripadajoče spletne storitve

Pripadajoča spletna storitev WSDL predstavlja klasičen zapis spletne storitve z vsemi potrebnimi informacijami. Poslovni proces jezika BPEL in pripadajoča spletna storitev WSDL sta med seboj povezani preko značke XML, imenovane »Import«. Povezava je potrebna predvsem s stališča podatkovnega modela, saj definira tip podatkov, ki bo uporabljen pri izvajanju poslovnega procesa. Za začetek izvajanja poslovnega procesa je potrebno proženje pripadajoče spletne storitve – odjemalca. V veliki večini primerov je ob sprožitvi odjemalca potrebno posredovati tudi določene vhodne podatke, ki dejansko predstavljajo vhodne podatke poslovnega procesa. Proženje le-tega je izvedeno s strani odjemalca, ki posreduje vhodne podatke in na ta način izvede začetek izvajanja poslovnega procesa. Struktura vhodnih podatkov je definirana na strani pripadajočega odjemalca, uporabljena pa tudi s strani poslovnega procesa. Poleg podatkovnega modela, ki je skupen tako spletni storitvi kot tudi poslovnemu procesu, so tu še določene druge nastavitve, ki so uporabljene med postopkom prebiranja poslovnega procesa. Izdelana rešitev tako predvideva vnos nabora različnih vrst podatkov, ki tvorijo neko zaključeno celoto, potrebno za začetek izvajanja poslovnega procesa.

Določeni podatki in njihove definicije, ki so uporabljeni v poslovnem procesu, so dejansko izhajali iz pripadajoče spletne storitve. Struktura podatkov je definirana s pomočjo predloge XSD, ki je običajno definirana kar znotraj same definicije datoteke WSDL. Gledano s stališča poslovnega procesa to pomeni, da se med aktivnostmi prenašajo podatki, strukturirani v obliki notacije XML. Pri določenih (kompleksnih) aktivnostih je taka struktura podatkov močno zaželena, saj je preverjanje pogojev in poizvedb na tak način močno olajšano. Tak primer predstavljajo pogoji v procesu jezika BPEL, ki so izraženi s pomočjo izraznega jezika (*Expression language*) [39]. Za potrebe pridobivanja podatkovnega modela pripadajoče spletne storitve smo izdelali poseben modul – »BPEL_WS«, ki skrbi za preslikavo spletne storitve v javanske objekte. Modul je izdelan s pomočjo javanske knjižnice JAX-WS (*Java API for XML Web Services*) [19], kjer smo preko nastavitvene datoteke POM in dodatnih vezav JAXB uporabili eno njenih glavnih funkcionalnosti – »wsimport«. Ta poskrbi za preslikavo definicije WSDL v javanske objekte in s tem avtomatsko kreira implementacijo opisane spletne storitve. Ob njenem kreiranju se avtomatsko kreirajo tudi določeni javanski razredi, ki predstavljajo vhodne oz. izhodne podatkovne strukture. Te podatkovne strukture smo nato z uporabo že omenjene knjižnice JAXB s pridom uporabljali za kreiranje potrebnih vhodnih oz. izhodnih podatkov ob izvajanju poslovnega procesa. Omenjen modul predstavlja pomembno točko pri celotnem izvajanju poslovnih procesov v obliki delovnih tokov, saj poskrbi za pravilno strukturiranje podatkov med celotnim procesom. Poleg same strukture podatkov s preslikavo pridobimo tudi nekaj nastavitvenih podatkov, kot so imenski prostor, privzeta predpona itd.

Poleg generiranega podatkovnega modela, ki služi za izmenjavo sporočil, smo potrebovali tudi dodatne informacije o pripadajoči spletni storitvi. V ta namen smo implementirali dodatni modul – »BPEL_Utilities«, ki je zaradi generičnosti in uporabe v različnih primerih pripadal skupini podpornih modulov. Primer pridobivanja podatkov s strani spletne storitve je prikazan na sliki 5.5.

```
public void loadWSDLformFS(String filePath)
{
	WSDLFactory factory;
	try
	{
		factory = WSDLFactory.newInstance();
		WSDLReader reader = factory.newWSDLReader();
		wsdlDefinition = factory.newDefinition();
		String wsdlPath = this.getFilenameFromResources(filePath);
		wsdlDefinition = reader.readWSDL(wsdlPath);
		portList = this.parsePort();
		parseLacationURI();
		parseOperation();

		if(wsdlDefinition != null)
		{
			wsdlQname = wsdlDefinition.getQName();
			if(wsdlQname != null)
			{
				serviceLocalpart =  wsdlQname.getLocalPart();
				serviceNamespaceUri = wsdlQname.getNamespaceURI();
				servicePrefix =
					wsdlDefinition.getPrefix(serviceNamespaceUri);
				messages = wsdlDefinition.getMessages();
			}
			else
			{
				serviceLocalpart = getLocalPortFromQName();
				serviceNamespaceUri =
						wsdlDefinition.getTargetNamespace();
				servicePrefix =
					wsdlDefinition.getPrefix(serviceNamespaceUri);
				messages = wsdlDefinition.getMessages();
			}
			servicePortName = this.getPortName();
			servicePortType = this.getPortType();
		}
	}
	catch (WSDLException e) {
		System.out.println("An error occured while parsing wsdl on uri: " +
				wsdlURI + " ; " + e.getMessage());
	}
}
```

Slika 5.5: Pridobivanje različnih podatkov o spletni storitvi, ki so bili potrebni za povezovanje s poslovnim procesom jezika BPEL.

Pri izdelavi modula »BPEL_Utilities« smo si pomagali z obstoječo javansko knjižnico WSDL4J [40], ki omogoča lažje kreiranje, prebiranje in rokovanje z definicijo WSDL. Tako smo s pomočjo omenjene knjižnice lažje prišli do potrebnih podatkov, kot so:

- nabor spremenljivk – konstrukt »Variables«,
- dodatne povezave na nivoju spletnih storitev – konstrukt »Import«,
- ime vrat pri definiciji WSDL – »portName«,
- ime operacije pri definiciji WSDL – »operation«.

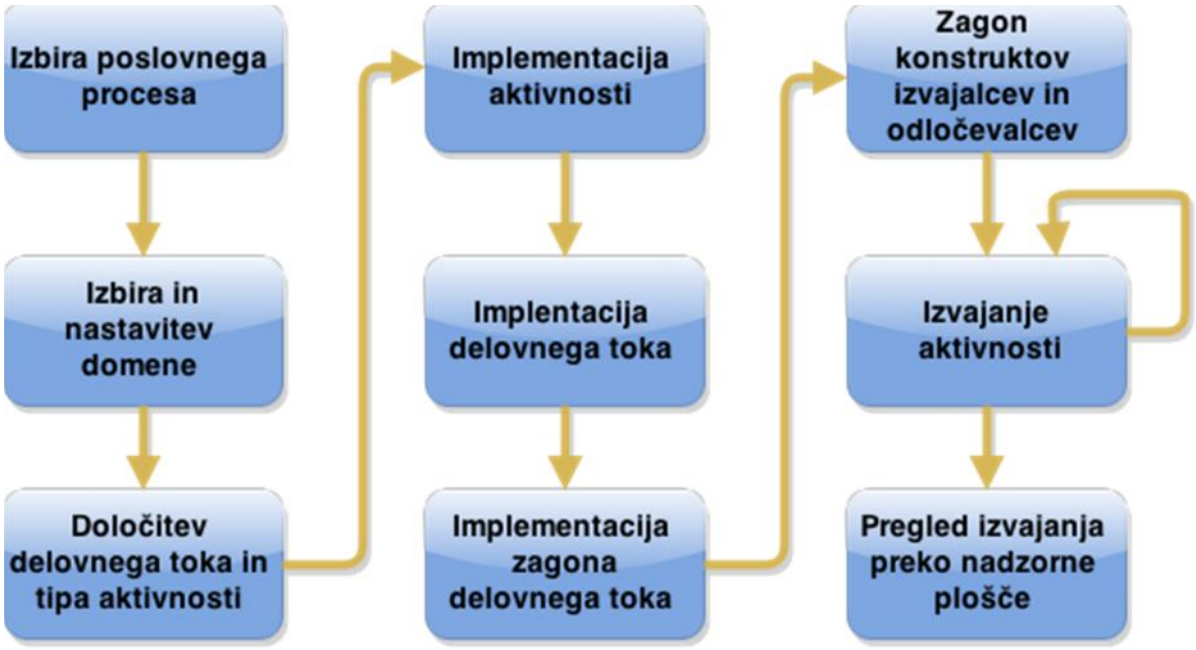
5.3 Poslovni proces v obliki delovnega toka spletne storitve SWF

Kot je značilno za vse spletne storitve, tudi spletne storitve podjetja Amazon delujejo na principu definiranih vmesnikov, preko katerih so njihove funkcionalnosti dosegljive uporabnikom po svetu. Enako velja za delovne tokove spletne storitve SWF, ki ponuja možnost izvajanja (in nadzora nad potekom) delovnih tokov. Za njihovo izvajanje obstajajo jasno definirani standardi in pravila, ki jih je potrebno upoštevati, če želimo izkoristiti vse ponujene prednosti. Struktura programske rešitve je zasnovana v obliki porazdeljene aplikacije, ki predvideva delitev posameznih implementacij delovnega toka na ločene razrede. Celotna rešitev je sestavljena iz več korakov, kot je prikazano na sliki 5.6:

- izbira poslovnega procesa,
- nastavitev domene,
- registracija aktivnosti in delovnega toka,
- implementacija aktivnosti,
- implementacija delovnega toka,
- izvedba delovnega toka in zagona,
- zagon izvajalcev,
- izvajanje aktivnosti in pregled,
- pregled izvedbe preko nadzorne plošče.

Izdelava funkcionalnosti si med seboj sledi v logičnem zaporedju. Tako je treba najprej izbrati želen poslovni proces in opraviti registracijo ter nastavitve domene. Na podlagi izbora poslovnega procesa sledi definiranje aktivnosti in registracija na spletni storitvi SWF. Ko je ta uspešno izvedena, opravimo implementacijo posameznih aktivnosti, kjer določimo poslovno logiko posamezne aktivnosti. Naslednji korak predstavlja registracija delovnega toka, kjer podamo nekaj osnovnih podatkov o delovnem toku – poslovnem procesu. Nato sledi implementacija delovnega toka, ki pri svojem izvajanju uporablja prej registrirane aktivnosti. Ko imamo registracije in implementacije zaključene, se lotimo še izvedbe delovnega toka. Izvedba poteka v fazah, kjer najprej registriramo konstrukte, tako imenovane delavce. Ti poskrbijo za izvajanje posameznih aktivnosti in delovnega toka. Po končani registraciji sledi proženje instance poslovnega procesa, kjer preko prej registriranih delavcev in potrebnih

vhodnih podatkov sprožimo zagon samega poslovnega procesa ter s tem tudi izvajanje prve aktivnosti. Nadzor nad izvajanjem delovnega toka in njegovih aktivnosti je omogočen preko nadzorne plošče.



Slika 5.6: Potek izvedbe delovnega toka na spletni storitvi SWF.

Kot pri vseh ostalih ponudnikih so tudi pri podjetju Amazon vse storitve plačljive. Za dostop do želenih storitev ne glede na tip je potrebno opraviti predhodno registracija uporabniškega računa. Na podlagi odobrene registracije dobimo tudi pristopne podatke, ki služijo za dostop do izbranih storitev. Enako velja za zgoraj opisani postopek izvajanja delovnega toka z uporabo spletne storitve SWF, kjer pred vsakim zagonom novega delovnega toka podamo svoje podatke v obliki dostopnih in skrivnih ključev. Na podlagi podanih podatkov se v ozadju opravi verifikacija in če imamo omogočene pravice, se sprožijo nadaljnji postopki za izvršitev želene zahteve.

5.3.1 Registracija aktivnosti in delovnega toka

Za uporabo aktivnosti v sklopu izvajanja delovnega toka spletne storitve SWF je potrebna predhodna registracija. Registracija posamezne aktivnosti je izvedena samo ob prvi pojavitvi slednje. V primeru spremembe definicije aktivnosti je treba izvesti novo registracijo. Za uspešno izvedbo le-te podamo nekaj osnovnih podatkov aktivnosti, kot so njeno ime, vhodni parametri in njena različica. Dodatno lahko ob registraciji podamo tudi parametre o določenih zakasnitvah izvedbe in posameznih nastavitvah, o normalizaciji ter denormalizaciji podatkov. Različica posamezne aktivnosti predstavlja spremembo definicije slednje. Posamezne aktivnosti je možno registrirati na nivoju regije, v kateri se določen delovni tok izvaja. Če zamenjamo regijo izvajanja delovnega toka, je potrebna ponovna registracija aktivnosti. Izvedba registracije je z uporabniškega vidika gledano avtomatska ob samem zagonu

delovnega toka. Vsaka registrirana aktivnost je definirana v obliki vmesnika z omenjenimi lastnostmi. Primer definicije vmesnika lahko vidimo na sliki 5.7, kjer opravimo registracijo določenih aktivnosti, definiranih z vhodnimi in izhodnimi podatki. Za dejansko uporabo aktivnosti pa je treba izdelati tudi njeno implementacijo. Implementacije posameznih aktivnosti se med seboj razlikujejo, saj so namenjene opravljanju različnih nalog. Tako smo opravili registracijo vseh relevantnih aktivnosti jezika BPEL, ki smo jih kasneje v sklopu izvajanja poslovnih procesov uporabljali pri izvedbi delovnih tokov spletne storitve SWF. Vsebina vsake izmed registriranih aktivnosti je bila odvisna od njenih nalog.

```
@ActivityRegistrationOptions(defaultTaskScheduleToStartTimeoutSeconds=300,
                             defaultTaskStartToCloseTimeoutSeconds=100)
@Activities(version="1.0", dataConverter=MyJsonDataConverter.class)
public interface BPELActivities
{
       public TActivity processInvoke(TActivity activity, Map<String, Object>
                    activityVariables, Map<String, Object> activityVariableValues);

       public TActivity processReply(TActivity activity, Map<String, Object>
                    activityVariables, Map<String, Object> activityVariableValues);

       public TSequence processIfElse(TActivity activity, Map<String, Object>
                    activityVariables, Map<String, Object> activityVariableValues);

       public List<TActivity> processWhile(TActivity activity,
                                Map<String, Object> activityVariables,
                                Map<String, Object> activityVariableValues);
```

Slika 5.7: Registracija in definicija vmesnika aktivnosti "Invoke", "Reply", "If" in "While".

Podobna pravila, kot veljajo pri registraciji aktivnosti, veljajo tudi pri registraciji delovnega toka. Tu (Slika 5.8) prav tako podamo nekaj osnovnih podatkov, kot so ime delovnega toka, definicija vhodnih podatkov in različica implementacije. Delovni tok smo tako kot vse aktivnosti definirali v obliki vmesnika. Za izvedbo samega delovnega toka smo dodali še njegovo implementacijo, kjer je bila slednja odvisna od vsebine poslovnega procesa. V našem primeru smo registrirali en delovni tok, ki je predstavljal generično implementacijo in pri tem podpiral izvedbo vseh relevantnih aktivnosti, ki so bile predhodno registrirane. Poleg glavnega toka smo izvedli tudi registracijo podtoka, ki smo jo uporabili zgolj v eksperimentalne namene.

```
@Workflow(dataConverter=MyJsonDataConverter.class)
@WorkflowRegistrationOptions(defaultExecutionStartToCloseTimeoutSeconds = 3600)
public interface BPELWorkflow {
	@Execute(version="1.0")
	public void ws_bpel(List<TActivity> activityList,
			Map<String, Object> variables,
			Map<String, Object> variableValues,
			Map<String, Object> additionalOptions);
}
```

Slika 5.8: Registracija in definicija vmesnika delovnega toka s pripadajočimi vhodnimi parametri.

5.3.2 Implementacija aktivnosti

Posamezne aktivnosti so med seboj neodvisne in vsaka predstavlja svojo enoto. Da bi posamezne aktivnosti združili v celoto, smo jih med seboj povezali z uporabo skupnih parametrov, kot so podatki o spremenljivkah in njenih tipih. Omenjeni podatki, ki so hranjeni v obliki dinamičnih struktur, so uporabljeni med celotnim poslovnim procesom in predstavljajo vhodne oz. izhodne podatke vsake posamezne aktivnosti. Na podlagi implementacije aktivnosti in njenih nalog je ob izvajanju slednjih prišlo do sprememb vrednosti spremenljivk. Te spremembe smo zabeležili v omenjene dinamične strukture, ki so hranile podatke o spremenljivkah uporabljenih, ob izvedbi aktivnosti. Rezultat izvedene aktivnosti so tako predstavljali podatki o spremenljivkah in njihovih vrednostih. Izvajanje aktivnosti v okviru delovnih tokov spletne storitve SWF je predstavljal tudi velik zalogaj v smislu izmenjave podatkov. Definicija posameznih aktivnosti je bila namreč zabeležena na spletni storitvi SWF, sama implementacija spletne storitve pa na gostiteljskem računalniku. Za uspešno izvajanje implementacije posamezne aktivnosti je bila potrebna neprestana komunikacija v oblik zahtevkov in odgovorov. Komunikacija je implementirana v obliki knjižnice programskega paketa Java, definiranega s strani podjetja Amazon. Vsebina, ki se izmenjuje pri komunikaciji, je bila definirana v obliki objektov, namenjenih prenosu podatkovnih objektov, zapisanih z uporabo odprtokodnega standarda JSON (*JavaScript Object Notation*). Tak format izmenjave podatkov je povzročal nemalo težav. Do problema pri sami strukturi podatkov JSON smo naleteli pri normalizaciji javanskih objektov, ki smo jih pridobili na podlagi notacije XML. Za potrebe normalizacije je bila uporabljena (s strani SDK) vgrajena javanska knjižnica (Jackson), ki je poskrbela za preslikavo med objekti JSON in javanskimi podatkovnimi strukturami. Težava je nastala pri določenih podatkovnih strukturah (seznam serializiranih objektov – List<Serializable>), pri katerih ni bila možna avtomatska normalizacija v format JSON. Za uspešno premostitev nastalega problema je bilo treba implementirati normalizacijo in tudi denormalizacijo po meri. V ta namen smo za vsak javanski objekt poleg privzete normalizacije oz. denormalizacije implementirali dodatno preslikavo v smislu prebiranja podatkov JSON v (za prebrano strukturo) primerne javanske

dinamične strukture (»ArrayList«, »Map« itd.) in obratno. Tak način normalizacije smo uporabljali predvsem pri aktivnosti »Assign« in njeni hierarhični strukturi konstruktov, ki jih zajema (»Copy«, »From«, »To«, »Literal, »Query«).

Določene aktivnosti so bile neposredno preslikane, zato ločena implementacija na nivoju delovnega toka ni bila potrebna. To velja predvsem za enostavne aktivnosti, ki niso spreminjale vrstnega reda izvajanja aktivnosti – »Receive«, »Reply«, »Invoke«, »Assign«. Za primere kompleksnih aktivnosti, kot so »If«, »While«, »RepeatUntil« in »ForEach«, smo potrebovali dodatno implementacija na nivoju delovnega toka. Vse aktivnosti, ki se izvajajo pri poteku delovnega toka, so izvedene v obliki asinhronih klicev. Tak način izvedbe je predviden s strani spletne storitve SWF, saj se vsaka izmed aktivnosti nahaja na oddaljeni lokaciji. Izvedba določene aktivnosti je odvisna od kompleksnosti njene implementacije, kar vpliva na čas izvedbe. Z uporabo asinhronih klicev pri izvedbi posameznih aktivnosti zagotovimo pravilen vrstni red izvajanja aktivnosti glede na zaporedje, definirano v poslovnem procesu jezika BPEL. Rezultat izvedbe delovnega toka predstavljajo pridobljene vrednosti, ki se prenašajo med posameznimi aktivnostmi.

Za potrebe izvedbe delovnega toka na spletni storitvi SWF smo v sklopu modula z imenom »SWF«, ki je podpiral samo izvedbo, izdelali tudi veliko pomožnih razredov, funkcij in različnih dinamičnih struktur, s katerimi smo podprli različne funkcionalnosti. Podporne razrede smo izdelali za potrebe določene aktivnosti (»Assign« – pomožne funkcije pri izvedbi prirejanja vrednosti spremenljivkam) ali za skupne potrebe več različnih aktivnosti (»If«, »While«, »RepeatUntil« – pomožne funkcije za evalvacijo pogojev, slika 5.9).

```
/**
 *     Pretvorba $input.payload (BPEL spremenljivka) v /tns:input etc
 *     /tns:input etc
 * */
private static String parseCondition(String condition) {
	String returnCondition = null;
	String regex = "(\\$\\w*\\.?\\w*/?)";
	Pattern p = Pattern.compile(regex,Pattern.CASE_INSENSITIVE |
					Pattern.DOTALL);
	Matcher m = p.matcher(condition);
	List<Object> resultList = new ArrayList<Object>();
	if (m.find())
	{
		for (int i = 1; i <= m.groupCount(); i++) {
			resultList.add(m.group(i));
		}
	}
	for (Object object : resultList) {
		returnCondition = condition.replace((CharSequence) object, "");
	}
	log.debug("Parsed input condition: " + condition +
			" and now returning new conditon: "+ returnCondition);
	return returnCondition;
}
```

Slika 5.9: Primer pomožne funkcije za luščenje spremenljivk procesa BPEL iz pogojnega stavka.

5.3.3 Odjemalec spletne storitve

Za potrebe izvajanja klicev na različne spletne storitve v okviru aktivnosti »Invoke« smo izdelali modul »WSClient«, ki je deloval kot generični odjemalec za poljubno spletno storitev. Sama implementacija modula je bila dokaj zahtevna, saj je deloval kot univerzalni odjemalec za poljubno spletno storitev. Glavna naloga izdelanega odjemalca je, da na podlagi vhodnih parametrov, ki so vsebovali informacije o spletni storitvi, pravilno kreira zahtevo (glede na izbrano spletno storitev) in jo posreduje v obliki zahteve na spletno storitev. Primer izvedbe klica z izsekom kode, ki poskrbi za izvedbo klica spletne storitve, je prikazan na sliki 5.10. Pri tem smo za potrebe dinamične implementacije uporabili tudi koncept »Java Reflection«, ki je nudil možnost pregledovanja in urejanja obnašanja programa v času izvajanja. Največje izzive so predstavljali identifikacija operacij in vrat ter kreiranje zahtev spletnih storitev na podlagi prejetih podatkov v obliki vhodnih parametrov. Kot smo že omenili v poglavju 5.2.3, je struktura vhodnih podatkov definirana na strani spletne storitve in je za vsako spletno storitev drugačna. Na tej točki smo izkoristili prednosti koncepta »Java Reflection«, s katerim smo na podlagi imen razredov (kreiranih iz definicij WSDL) in ostalih podatkov, ki so bili dinamično pridobljeni, uspešno identificirali pripadajoče spletne storitve. Poleg identifikacije smo poskrbeli tudi za kreiranje pravilne strukture podatkov, ki so predstavljali vhodne parametre, saj so bili odvisni od strukture podatkov, definiranih s strani

spletne storitve. Pri implementaciji smo si pomagali z uporabo funkcionalnosti iz predhodno izdelanih podpornih modulov(»BPEL_Utilities«). Zaradi kompleksnosti in potrebe po uporabi na več mestih smo izdelali funkcionalnost v obliki sklopljene celote in jo zapisali v ločen modul.

```
if(portMethodName.startsWith("get") &&
portMethodName.contains(this.wsdlFP.getServicePortName())) {
	portObj = portMethod.invoke(service, noparams);
	log.info("Successfully invoked method: " + portMethodName);
	for(Method operationMethod: portObj.getClass().getDeclaredMethods()) {
	//	po uspešnem klicu operacije get* preverjanje ostalih klicev funkcije
	//	get*
		String operationMethodName = operationMethod.getName();
		if(operationMethodName.startsWith("get") &&
			operationMethodName.compareTo(this.operation) == 0) {
		//	kreiranje parametrov
			Object paramObj =
			createInputParameter(operationMethod.getParameterTypes(),
						this.input);
			if(paramObj == null) {
				log.error("An error occured when creating input
						paramters resulting with null.");
				break;
			}
			//	izvedba klica funkcije s pripravljenimi parametri
			operationObj = operationMethod.invoke(portObj, paramObj);
			log.info("Successfully invoked method: " +
				operationMethodName);
			//	pridobivanje tipa vrnjenega razreda in kreiranje nove
			//	instance
			Class<?> returnClass =
						getClass(operationMethod.getReturnType());
			//	preverjanje vseh funkcij get*
			for(Method returnMethod: returnClass.getDeclaredMethods()) {
				String returnMethodName = returnMethod.getName();
				if(returnMethodName.startsWith("get")) {
					Object returnObj =
					returnMethod.invoke(operationObj, noparams);
					log.info("Successfully invoked method: " +
							returnMethodName);
					return returnObj;
				}
			}
		}
	}
break;
}
```

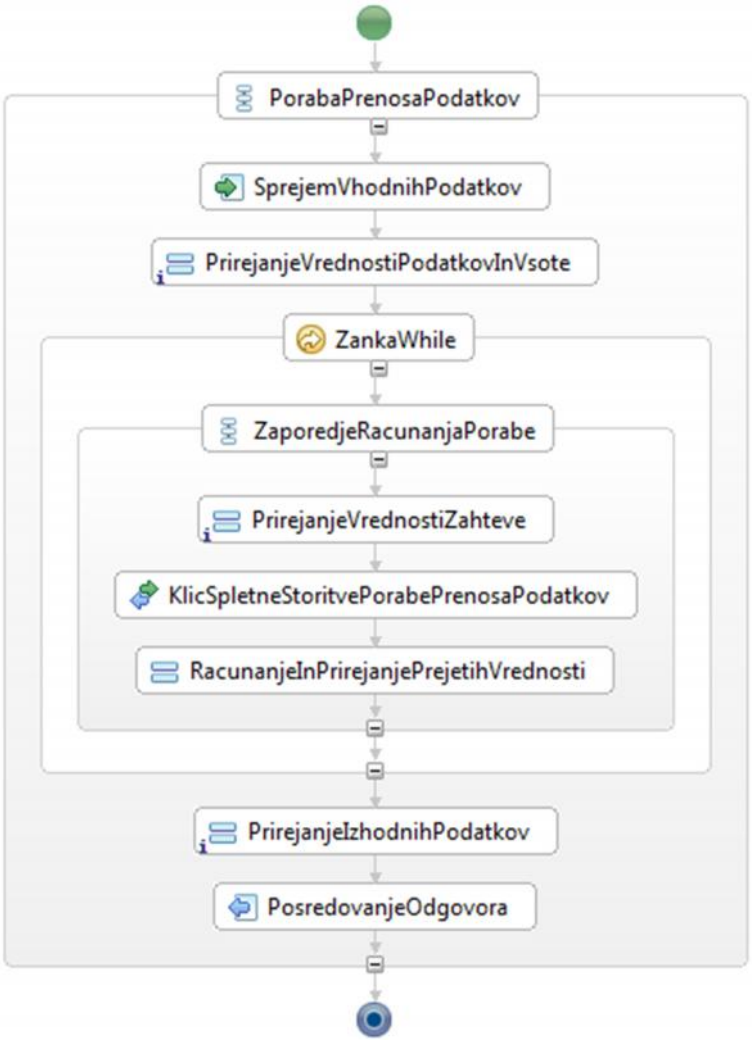
Slika 5.10: Dinamična izvedba klica spletne storitve na podlagi vhodnih parametrov.

5.4 Izvedba poslovnega procesa v obliki delovnih tokov spletne storitve SWF

Za izvedbo poslovnega procesa jezika BPEL v obliki delovnih tokov spletne storitve SWF smo izdelali nabor testnih poslovnih procesov, ki so podrobneje opisani v šestem poglavju in so sestavljeni iz več različnih aktivnosti. Pri izbiri aktivnosti omenjenih poslovnih procesov smo se osredotočili na tiste, ki so najpogosteje v uporabi. Kot primer izvedbe smo izbrali poslovni proces, ki je podpiral izračun porabe storitve prenosa podatkov mobilne naprave za določeno časovno obdobje.
Poslovni proces je sestavljen iz naslednjih aktivnosti:

- »Receive«,
- »Assign«,
- »While«,
- »Invoke«,
- »Reply«.

Vsebina poslovnega procesa predstavlja funkcionalnost preverjanja porabe storitve prenosa podatkov za podano časovno obdobje. V našem primeru je obdobje za izračun podano v obliki števila preteklih dni. Za potrebe testiranja smo izdelali tudi spletno storitev, ki je na podlagi zahteve podala statične podatke o porabi za določen dan. Proženje začetka izvajanja poslovnega procesa je pogojeno z vnosom vhodnih podatkov s strani uporabnika. Slednji je kot vhodni podatek podal želeno število preteklih dni. Na podlagi vnosa vhodnega podatka se je pričela izvedba poslovnega procesa, prikazanega na sliki 5.11.



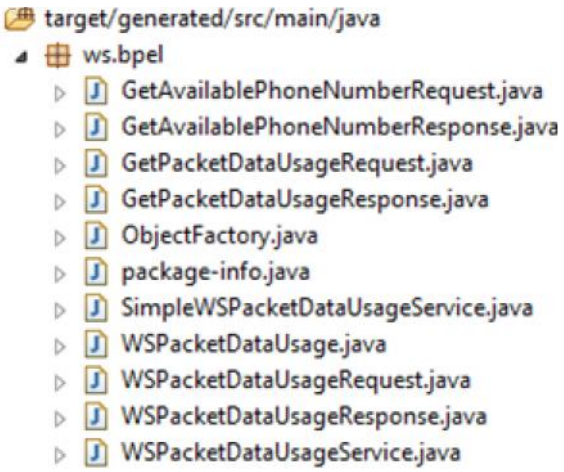
Slika 5.11: Testni primer poslovnega procesa, ki podpira funkcionalnost izračuna porabe storitve prenosa podatkov za podano časovno obdobje.

V prvem koraku se opravi sprejem vrednosti vhodnih podatkov v obliki aktivnosti »Receive« (»SprejemVhodnihPodatkov«). Sledi prirejanje vrednosti vhodnih podatkov in inicializacija pomožnih spremenljivk (za potrebe računanje vsote) v obliki aktivnosti »Assign« (»PrirejanjeVrednostiPodatkovInVsote«). Po končani inicializaciji smo začeli z izvajanjem zanke, implementirane v obliki aktivnosti »While« (»ZankaWhile«), ki je v iteracijah izvajala zaporedje aktivnosti. Pogoj omenjene zanke predstavlja vhodni podatek o številu dni. Zaporedje aktivnosti je podprto z uporabo aktivnosti »Sequence«, ki vsebuje gnezdene aktivnosti »Assign« in »Invoke«. Prva gnezdena aktivnost tipa »Assign« (»PrirejanjeVrednostiZahteve«) skrbi za inicializacijo in luščenje vhodnih podatkov v format zahteve spletne storitve. Sledi izvedba klica spletne storitve v obliki aktivnosti »Invoke« (»KlicSpletneStoritvePorabePrenosaPodatkov«), ki izvede klic zunanje spletne storitve na podlagi predhodno pripravljenih podatkov. Na podlagi odgovora spletne storitve opravimo izračun vsote porabe. Rezultat shranimo v potrebne spremenljivke in povečamo vrednost

števca v okviru druge izmed aktivnosti »Assign« (»RacunanjeInPrirejanjePrejetihVrednosti«). Na podlagi primerjanja števca (ki se ob vsaki iteraciji povečuje) in vhodnega podatka o številu dni, se zanka ponavlja, medtem ko je pogoj izpolnjen. Po končanem izvajanju zanke sledi prirejanje izračunanih vrednosti izhodnim spremenljivkam v obliki aktivnosti »Assign« (»PrirejanjeIzhodnihPodatkov«). Prirejeni podatki se nato uporabijo v obliki spremenljivke pri izvedbi aktivnosti »Reply« (»PosredovanjeOdgovora), ki predstavlja končno aktivnost poslovnega procesa.

Prototipna aplikacija za izvedbo opisanega poslovnega procesa v obliki delovnih tokov spletne storitve SWF potrebuje vhodne podatke. Ti so posredovani v obliki poslovnega procesa jezika BPEL in pripadajoče spletne storitve (v obliki definicije WSDL). Definicijo WSDL v obliki dokumenta odložimo na določeno lokacijo, ki je definirana v sklopu nastavitev modula »WS_BPEL«. Na podlagi vhodnih parametrov kreiramo potrebne razrede pripadajoče spletne storitve (Slika 5.12), ki predstavljajo potrebne podatkovne strukture pripadajočega odjemalca.

```
<groupId>org.codehaus.mojo</groupId>
<artifactId>jaxws-maven-plugin</artifactId>
<version>1.12</version>
<executions>
  <execution>
    <id>WS_BPELWSDL</id>
      <goals><goal>wsimport</goal></goals>
         <configuration>
           <!-- Direktorij datotek WSDL -->
            <wsdlDirectory>
             src/main/resources/wsdl
            </wsdlDirectory>
            <!-- datoteka WSDL -->
            <wsdlFiles>
               <wsdlFile>
            WS_AvailablePhoneNumberArtifacts.wsdl
               </wsdlFile>
            </wsdlFiles>
            <keep>true</keep>
            <!-- Ime paketa -->
            <packageName>ws.bpel</packageName>
            <!-- hranjenje generiranih datotek -->
            <sourceDestDir>target/generated/src/main/java</sourceDestDir>
         </configuration>
  </execution>
</executions>
```



Slika 5.12: Nastavitvena datoteka POM, ki generira razrede, pripadajoče spletne storitve na podlagi definicije WSDL.

Sledi izvedba poslovnega procesa z uporabo modula »BPELObjectMapper«. Ta na podlagi nastavitvene datoteke prebere predhodno podane vhodne podatke v obliki dveh dokumentov (poslovni proces jezika BPEL in definicija WSDL). Omenjena dokumenta odložimo na

določeno lokacijo, definirano v sklopu nastavitev modula. Preko razreda, ki skrbi za zagon izvedbe poslovnega procesa, zaženemo novo instanco izvedbe delovnega toka in pri tem podamo potrebne vhodne parametre. Vhodni parametri vsebujejo podatke o aktivnostih poslovnega procesa, njihovih spremenljivkah, vrednostih in dodanih nastavitvah (Slika 5.13).

```
BPELMain swf = new BPELMain();
swf.startSWF(this.processor.getActivityList(),variables,variableValues,options);
```

Slika 5.13: Proženje izvedbe poslovnega procesa z vsemi pripadajočimi vhodnimi podatki.

V našem primeru vlogo pripadajočega odjemalca namesto spletne storitve prevzame v te namene izdelan javanski razred. Slednji poskrbi za kreiranje potrebnih vhodnih parametrov – seznam aktivnosti poslovnega procesa, njegove spremenljivke in njihove vrednosti (Slika 5.14) ter nastavitve. Od tu naprej izvedbo delovnega toka prevzame modul istoimenske spletne storitve SWF. Ta poskrbi za začetek izvedbe poslovnega procesa s tem, da izvede prijavo na spletno storitv SWF (podatki o dostopnem in skritem ključu), izbiro domene, lokacije strežnikov in kreira instance odjemalcev za komunikacijo s spletno storitvijo. Po uspešno opravljeni prijavi in inicializaciji odjemalca se izvede še registracija konstruktov delavcev za potrebe aktivnosti ter delovnega toka. Sledi začetek izvajanja delovnega toka, ki preko implementacije poslovnega procesa izvaja aktivnosti, definirane v obliki vmesnikov. Med izvajanjem delovnega toka spletna storitev SWF komunicira z gostiteljskim računalnikom, kjer je podana dejanska implementacija posameznih aktivnosti.

Variables: {
PacketDataUsagePLRequest=my.ws.client.GetPacketDataUsageRequest,
iterator=java.lang.Object,
input=ws.bpel.WS_PacketDataUsageRequest,
default=java.lang.Object,
sumValue=java.lang.Object,
PacketDataUsagePLResponse=my.ws.client.GetPacketDataUsageResponse,
output=ws.bpel.WS_PacketDataUsageResponse,
nrOfDays=java.lang.Object }
VariableValues: {
PacketDataUsagePLRequest=

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
      <getPacketDataUsageRequest
      xmlns="http://my.ws.mobile.operator.service/">
            <usageLastDays></usageLastDays>
      </getPacketDataUsageRequest>,
```

input=

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
      <ns2:WS_PacketDataUsageRequest
            xmlns:ns2="http://my.ws.mobile.operator.service"
            xmlns="http://my.ws.mobile.operator.service/">
            <ns2:input>2</ns2:input>
      </ns2:WS_PacketDataUsageRequest>,
```

iterator=,
default=

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><default/>,
```

PacketDataUsagePLResponse=

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
      <getPacketDataUsageResponse
            xmlns="http://my.ws.mobile.operator.service/">
            <usageDayResult></usageDayResult>
      </getPacketDataUsageResponse>,
```

sumValue=,
output=

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
      <ns2:WS_PacketDataUsageResponse
            xmlns:ns2="http://my.ws.mobile.operator.service"
            xmlns="http://my.ws.mobile.operator.service/">
            <ns2:result></ns2:result>
      </ns2:WS_PacketDataUsageResponse>,
```

nrOfDays= }

Slika 5.14: Generirani vhodni podatki, pridobljeni iz definicije WSDL in procesa jezika BPEL.

Postopek poteka izvedbe delovnega toka je viden preko nadzorne plošče spletne storitve SWF, kot prikazuje slika 5.15. Z uporabo nadzorne plošče pridobimo podatke o delovnem toku, vseh izvedenih aktivnostih in njihovem vrstnem redu, pa tudi o toku podatkov, ki se med izvajanjem izmenjujejo med delovnim tokom in aktivnostmi.

Workflow Execution: 01a37260-9b13-4caf-abb6-555a46c58360

Domain: BPEL2SWF

Summary | Events | **Activities**

Activi	Name	Versio	Status	Schedule Time	Start Time
34	BPELActivities.getVariableValues	1.0	Completed	Sunday, April 5, 2015 3:58:08 PM UTC+2	Sunday, April 5, 2015 3:58:08 PM UTC+2
35	BPELActivities.getAdditionalOptions	1.0	Completed	Sunday, April 5, 2015 3:58:08 PM UTC+2	Sunday, April 5, 2015 3:58:09 PM UTC+2
30	BPELActivities.getVariables	1.0	Completed	Sunday, April 5, 2015 3:58:05 PM UTC+2	Sunday, April 5, 2015 3:58:05 PM UTC+2
31	BPELActivities.getVariableValues	1.0	Completed	Sunday, April 5, 2015 3:58:05 PM UTC+2	Sunday, April 5, 2015 3:58:05 PM UTC+2
29	BPELActivities.processAssign	1.0	Completed	Sunday, April 5, 2015 3:58:03 PM UTC+2	Sunday, April 5, 2015 3:58:03 PM UTC+2
27	BPELActivities.getVariables	1.0	Completed	Sunday, April 5, 2015 3:58:01 PM UTC+2	Sunday, April 5, 2015 3:58:02 PM UTC+2
28	BPELActivities.getVariableValues	1.0	Completed	Sunday, April 5, 2015 3:58:01 PM UTC+2	Sunday, April 5, 2015 3:58:02 PM UTC+2
26	BPELActivities.processInvoke	1.0	Completed	Sunday, April 5, 2015 3:57:59 PM UTC+2	Sunday, April 5, 2015 3:57:59 PM UTC+2
24	BPELActivities.getVariables	1.0	Completed	Sunday, April 5, 2015 3:57:56 PM UTC+2	Sunday, April 5, 2015 3:57:56 PM UTC+2
25	BPELActivities.getVariableValues	1.0	Completed	Sunday, April 5, 2015 3:57:56 PM UTC+2	Sunday, April 5, 2015 3:57:56 PM UTC+2
23	BPELActivities.processAssign	1.0	Completed	Sunday, April 5, 2015 3:57:53 PM UTC+2	Sunday, April 5, 2015 3:57:53 PM UTC+2

Slika 5.15: Nadzorna plošča in prikaz izvedenih aktivnosti testnega poslovnega procesa jezika BPEL.

Končni rezultat je uporabniku predstavljen v obliki vrednosti spremenljivk aktivnosti »Reply«, ki je posredovana odjemalcu v obliki izpisa v konzoli. Rezultat testnega poslovnega procesa je prikazan na sliki 5.16.

[SWF Activity BPELActivityWorkerList 1] INFO swf.activity.BPELActivitiesImpl.processReply():
Started processing REPLY activity
[SWF Activity BPELActivityWorkerList 1] INFO swf.activity.BPELActivitiesImpl.processReply():
REPLY activity: 'replyOutput' finished successfully.
[SWF Activity BPELActivityWorkerList 1] INFO swf.activity.BPELActivitiesImpl.processReply():
Reply activity VariableValuesHolder:

{
PacketDataUsagePLRequest=

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
      <getPacketDataUsageRequest xmlns="http://my.ws.mobile.operator.service/">
            <usageLastDays>1</usageLastDays>
      </getPacketDataUsageRequest>,
```

input=

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
      <ns2:WS_PacketDataUsageRequest
                  xmlns:ns2="http://my.ws.mobile.operator.service"
                  xmlns="http://my.ws.mobile.operator.service/">
            <ns2:input>2</ns2:input>
      </ns2:WS_PacketDataUsageRequest>,
```

iterator=2,
default=<?xml version="1.0" encoding="UTF-8" standalone="yes"?><default/>,
PacketDataUsagePLResponse=

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
      <getPacketDataUsageResponse xmlns="http://my.ws.mobile.operator.service/">
            <usageDayResult>123</usageDayResult>
      </getPacketDataUsageResponse>,
```

sumValue=247,
output=

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
      <ns2:WS_PacketDataUsageResponse
                  xmlns:ns2="http://my.ws.mobile.operator.service"
                  xmlns="http://my.ws.mobile.operator.service/">
            <ns2:result>247</ns2:result>
      </ns2:WS_PacketDataUsageResponse>,
```

nrOfDays=2
}
2015-03-24 20:06:13,579 [SWF Activity BPELActivityWorkerList 1] INFO swf.activity.BPELActivitiesImpl.processReply() - **Reply activity VariableHolder**: { *PacketDataUsagePLRequest*=my.ws.client.GetPacketDataUsageRequest, *iterator*=java.lang.Object, *input*=ws.bpel.WS_PacketDataUsageRequest, *default*=java.lang.Object, *sumValue*=java.lang.Object, *PacketDataUsagePLResponse*=my.ws.client.GetPacketDataUsageResponse, *output*=ws.bpel.WS_PacketDataUsageResponse, *nrOfDays*=java.lang.Object }

Slika 5.16: Rezultat izvedbe poslovnega procesa jezika BPEL v obliki delovnega toka spletne storitve SWF v obliki izpisa v konzoli.

6 Verifikacija

Izvajanje poslovnih procesov temelji na naboru določenih pravil, ki definirajo postopek izvedbe poslovnega procesa glede na definirane pogoje. Pravila izvedbe definirajo poslovne zahteve določenega procesa, na podlagi katerih dosežemo želen rezultat. Ta pravila so običajno definirana v obliki zaporedja izvajanja določenih aktivnosti, ki predstavljajo nek poslovni proces. Vrstni red in tip aktivnosti je odvisen od vsebine poslovnega procesa, ki ga želimo realizirati. Enako velja za robne pogoje in njihovo vsebino, ki so predstavljeni v obliki poslovnih zahtev, s katerimi dosežemo veljaven rezultat. Preverjanje pravilnosti izvedbe poslovnega procesa in končnega rezultata je izvedena s primerjavo pravil poslovnih zahtev. Poslovni proces označimo za veljaven oz. pravilen, če se rezultat poslovnega procesa ujema s predvidenim rezultatom, določenim s strani poslovnih zahtev. Preverjanje pravilnosti ne temelji izključno na preverjanju končnega rezultata, saj je poleg tega prisotnih veliko drugih faktorjev, s katerimi potrdimo ustreznost izvedenega poslovnega procesa. Eden takih je tudi preverjanje poti izvedbe poslovnega procesa oz. preverjanje zaporedja izvajanja aktivnosti. Pri tem velja, da je poslovni proces izveden pravilno, če so izvedene vse aktivnosti v predvidenem vrstnem redu. Vrstni red izvedbe posameznih aktivnosti lahko variira v odvisnosti od poslovnih pogojev, definiranih znotraj samega procesa. V tem primeru lahko za različne vhodne podatke pridobimo različne rezultate in prav tako tudi različno zaporedje izvajanja aktivnosti. Pri vsakem poslovnem procesu je treba identificirati vse različice izvedbenih poti in jih tudi verificirati ter rezultate primerjati s pričakovanimi. Izvajanje različnih poti izvedb poslovnega procesa lahko dosežemo z različnimi vhodnimi podatki, pri tem pa se spreminjajo tudi izhodni podatki oz. rezultati, saj so le-ti odvisni od zaporedja aktivnosti. Posebno pozornost moramo posvetiti tudi dinamičnim pogojem, ki vsebujejo spremenljivke, saj so odvisni od določenih spremenljivk in njihovih vrednosti v določenem koraku poslovnega procesa. Nekatere izmed aktivnosti lahko s svojim delovanjem močno vplivajo na sam rezultat izvedbe poslovnega procesa. Ena izmed takih je tudi aktivnost »Invoke«, s katero izvedemo klic na zunanjo spletno storitev. V primeru neuspešnega klica oz. v primeru prejetja neveljavnih podatkov v obliki odgovora zunanje spletne storitve lahko to slabo vpliva na nadaljnji potek in izvedbo poslovnega procesa. Zaključek poslovnega procesa je v tem primeru lahko neuspešen, ali celo prekinjen. Take dogodke lahko preprečimo z uporabo mehanizmov za zaznavanje napak in sprožitvijo kompenzacij, ki poskrbijo za nadzorovano izvedbo poslovnega procesa v takih primerih.

6.1 Verifikacija aktivnosti

Definiranje pravil in izvedbe poslovnega procesa predstavlja osnovo za uspešno izvajanje poslovnega procesa. Če se želimo izogniti neželenim situacijam, je treba dobro definirati vsak poslovni proces in v postopku izvajanja nato spremljati izvedbo vsake njegove aktivnosti. Pri preslikavi poslovnih procesov na delovne tokove spletne storitve SWF smo uporabili podoben

pristop kot pri preverjanju izvajanja poslovnih procesov. Samo verifikacijo smo izvedli nad določenim naborom aktivnosti na več nivojih. Prvi nivo je predstavljal verifikacijo posameznih aktivnosti. Za vsako izbrano aktivnost smo kreirali primer primitivnega poslovnega procesa, kjer je jedro predstavljala izbrana aktivnost. Tak poslovni proces ni bil funkcionalno uporaben, služil je zgolj za verifikacijo pravilnosti izvajanja posamezne aktivnosti. Ker pa posamezna aktivnost sama po sebi ne more predstavljati poslovnega procesa, smo za potrebe preverjanja poslovni proces opremili le z osnovnim naborom zahtevanih aktivnosti. Tako smo vsak tak poslovni proces sestavili iz povprečno treh aktivnosti s poudarkom na aktivnosti, ki smo jo verificirali. Tak poslovni proces smo najprej analizirali in določili njegove robne pogoje in predvideli določen nabor vhodnih podatkov ter definirali predvidene izhodne podatke oz. rezultate. Na podlagi definicije pravil smo izvedli poslovni proces na razvojnem okolju, ki je predstavljal testno orodje za izvedbo BPM in pri tem opravili simulacijo poslovnega procesa. Sledila je izvedba simuliranega poslovnega procesa v obliki delovnih tokov spletne storitve SWF, kjer smo na podlagi definiranih vhodnih podatkov s pomočjo nadzorne plošče spletne storitve SWF preverjali zaporedje izvajanja aktivnosti. Poleg zaporedja smo sledili tudi toku podatkov, ki se je pretakal med aktivnostmi. Pri aktivnostih, ki so vsebovale več različnih izvedbenih poti glede na vhodne podatke ali dinamične pogoje, smo postopek verifikacije z različnim naborom podatkov večkrat ponovili. Pri tem smo uporabili primerne podatke, s katerimi smo preverili vse različice izvedbenih poti. Za uspešen zaključek verifikacije posamezne aktivnosti smo preverili še izhodni rezultat glede na vhodne podatke. Verifikacijo smo označili za uspešno, če so se dobljeni podatki o zaporedju izvedbe posameznih aktivnosti in končnih rezultatih, ujemajo z rezultati simulacije poslovnega procesa. S tem smo dobili verificiran nabor posameznih aktivnosti, ki so bile primerne za uporabo v procesih iz realnega sveta.

Drug nivo verifikacije je predstavljal izvedbo poslovnih procesov na primerih iz realnega sveta. Za potrebe verificiranja smo izdelali štiri primere poslovnih procesov iz različnih panog, ki so v poslovnem svetu pogosto v uporabi. Pri vseh štirih izbranih poslovnih procesih je veljalo pravilo, da so vsebovali aktivnosti iz nabora predhodno verificiranih aktivnosti. Vsak izmed štirih poslovnih procesov je vseboval različen nabor aktivnosti glede na poslovne zahteve. Aktivnosti so se na podlagi poslovne logike pojavljale v različnem zaporedju in se pri tem med seboj različno povezovale. Na ta način smo zagotovili pravilnost delovanja posamezne aktivnosti v različnih okoliščinah in obenem izvedli verifikacijo več medsebojno povezanih aktivnosti. Za razliko od prvega nivoja verifikacije, kjer smo preverjali pravilnost delovanja posamezne aktivnosti, smo tu izvedli verifikacijo celotnega poslovnega procesa. Tega smo podobno kot na prvem nivoju predhodno analizirali, določili vse izvedbene poti ter vhodne in izhodne podatke. Poleg analize smo izvedli tudi simulacijo na razvojnem okolju ter popisali ugotovitve in rezultate. Sledila je izvedba poslovnega procesa v obliki delovnega toka spletne storitve SWF. Pri tem smo preverjali vrstni red izvajanja posameznih med seboj

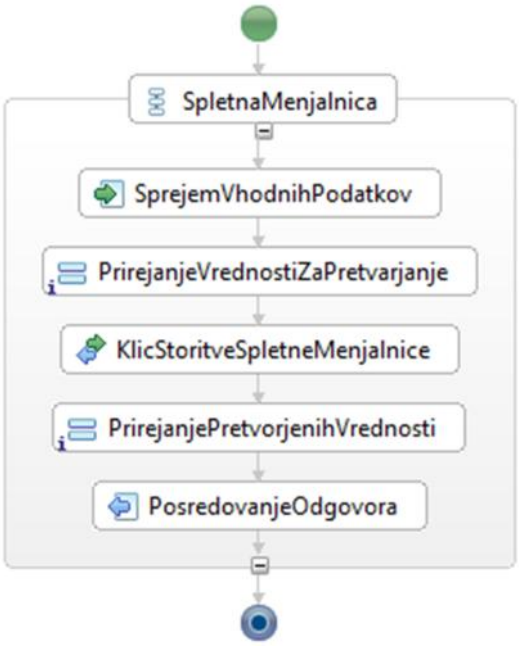
povezanih aktivnostih, pri različnih vhodnih podatkih ter končni rezultat izvedbe ob zaključku poslovnega procesa. Posamezni poslovni proces smo izvedli večkrat z različnim naborom vhodnih podatkov in pri tem dosegli izvajanje različne aktivnosti. Rezultate izvedbe smo primerjali s predhodno definiranimi in ovrednotenimi rezultati, ki smo jih pridobili na podlagi simulacije. Če so se ujemali, smo poslovni proces označili kot uspešno izveden.

6.2 Testni scenariji in metrike

Poleg preverjanja uspešnosti izvedbe posameznih poslovnih procesov smo izvedli tudi verifikacijo z uporabo za to predvidenih metrik. S tem smo želeli primerjati razlike pri izvedbi posameznih aktivnosti na testnem okolju jezika BPEL in v obliki delovnih tokov spletne storitve SWF. Pri tem smo izhajali iz predpostavke, da s preslikavo poslovnih procesov jezika BPEL na delovne tokove spletne storitve SWF ne povečamo kompleksnost poslovnega procesa. Z uporabo metrik smo izvedli primerjavo kompleksnosti poslovnega procesa in delovnega toka. Tradicionalno ovrednotenje z uporabo metrik o številu vrstic napisane kode (*LOC – Lines of code*) ni prilagojeno za potrebe poslovnih procesov [9], zato smo ovrednotenje zamenjali z metrikami, ki temeljijo na aktivnostih:

- metrika števila aktivnosti v procesu (*NOA – Number of activities*) [4], s pomočjo katere ovrednotimo poslovni proces glede na število aktivnosti,
- metrika števila aktivnosti in kontrolnih tokov v procesu (*NOAC – Number of activities and control-flow elements*) [4], ki poleg osnovnih aktivnosti, zajetih z metriko NOA, ovrednoti tudi aktivnosti v kontrolnih tokovih,
- metrika, ki ovrednoti število kontrolnih poti med procesom (*MCC – McCabe cyclometric complexity metric*) [14] in je definirana s formulo e - n +2, kjer e predstavlja število povezav, n pa število vozlišč (aktivnosti),
- metrika, ki ovrednoti število kontrolnih poti z različnimi prehodi (*CFC – Control-flow complexity metric*) [14] in temelji na MCC metriki ter upošteva različne prehode glede na pogoje (XOR, OR, AND).

Prvi testni primer poslovnega procesa podpira funkcionalnost spletne menjalnice. Gre za klasičen primer menjalnice, kjer se na podlagi vhodnih podatkov, ki vsebujejo trenutno in menjalno valuto ter vsoto, opravi klic zunanje spletne storitve. Slednja odgovori z zamenjano vsoto v želeni valuti. Primer poslovnega procesa jezika BPEL je prikazan na sliki 6.1.



Slika 6.1: Poslovni proces, ki podpira funkcionalnost spletne menjalnice.

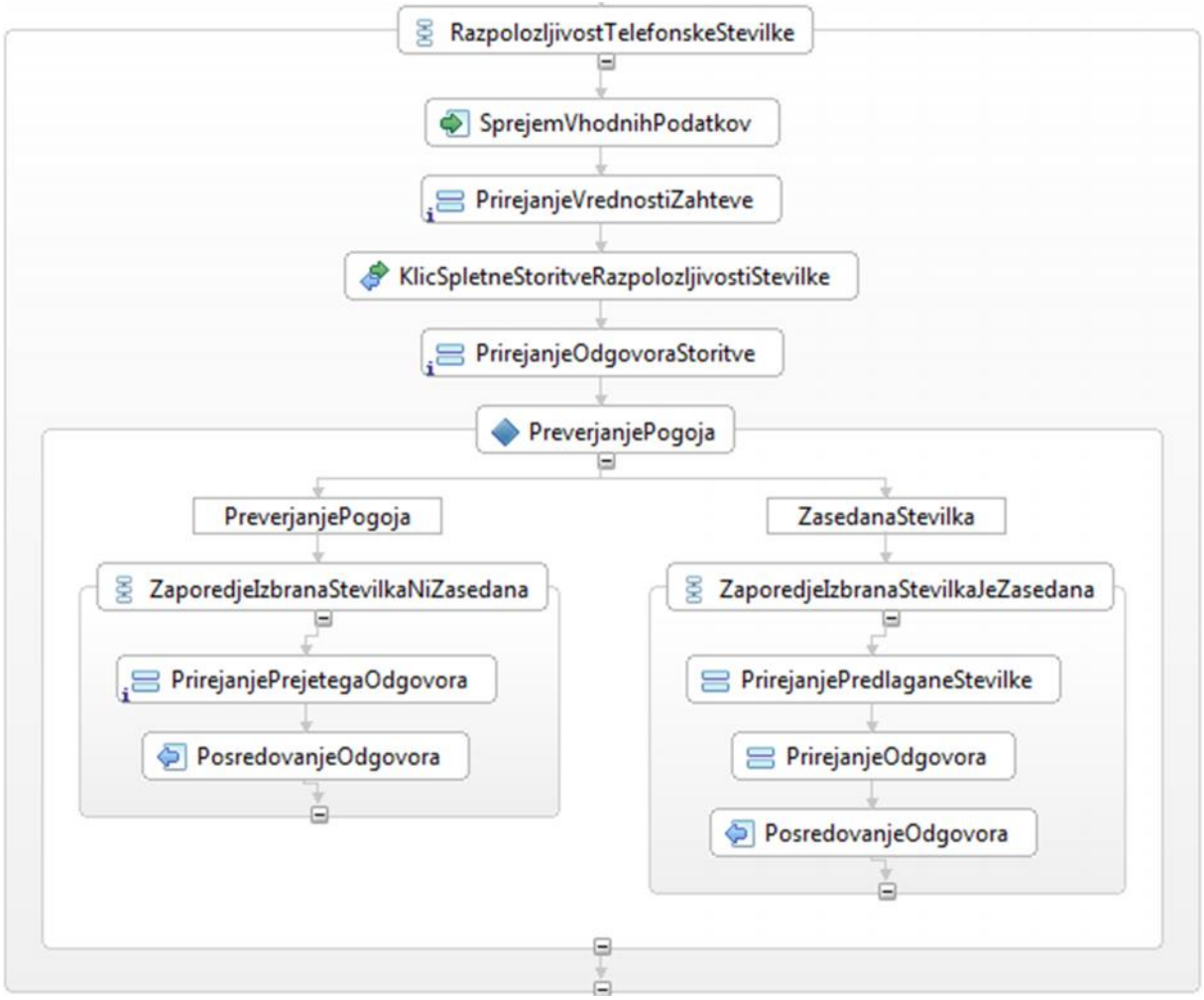
Primerjava med procesom, izvedenim na razvojnem okolju, in delovnim tokom, izvedenem preko spletne storitve SWF, ne pokaže bistvenih razlik. Primerjava posameznih metrik je predstavljena v tabeli 6.1. Razlike med posameznimi različnimi metrikami niso vidne, saj poslovni proces temelji na relativno enostavnih aktivnostih. V primeru obravnavanega poslovnega procesa ne uporabljamo nobene vejitve ali pogojno ponavljajočih se segmentov. Rezultat primerjave nam pove, da se poslovni proces v celoti izvede na povsem enak način na obeh okoljih.

Metrika	**Proces jezika BPEL v testnem okolju**	**Proces jezika BPEL preslikan na delovne tokove spletne storitve SWF**	**Razlika v %**
NOA	5	5	0
NOAC	0	0	0
MCC	2	2	0
CFC	0	0	0

Tabela 6.1: Primerjava metrik med procesom jezika BPEL in delovnih tokov spletne storitve SWF na primeru spletne menjalnice.

Drugačne rezultate dobimo pri poslovnem procesu, prikazanemu na sliki 6.2, ki podpira funkcionalnost preverjanja razpoložljivosti določene telefonske številke ob zakupu pri enem od ponudnikov storitev mobilne telefonije. Gre za poslovni proces, ki na podlagi vnesene telefonske številke preko klica zunanje spletne storitve opravi preverjanje razpoložljivosti le-te. Rezultat preverjanja je uporabniku posredovan v obliki odgovora zunanje spletne storitve,

kjer v primeru zasedenosti podane telefonske številke predlaga registracijo nove.



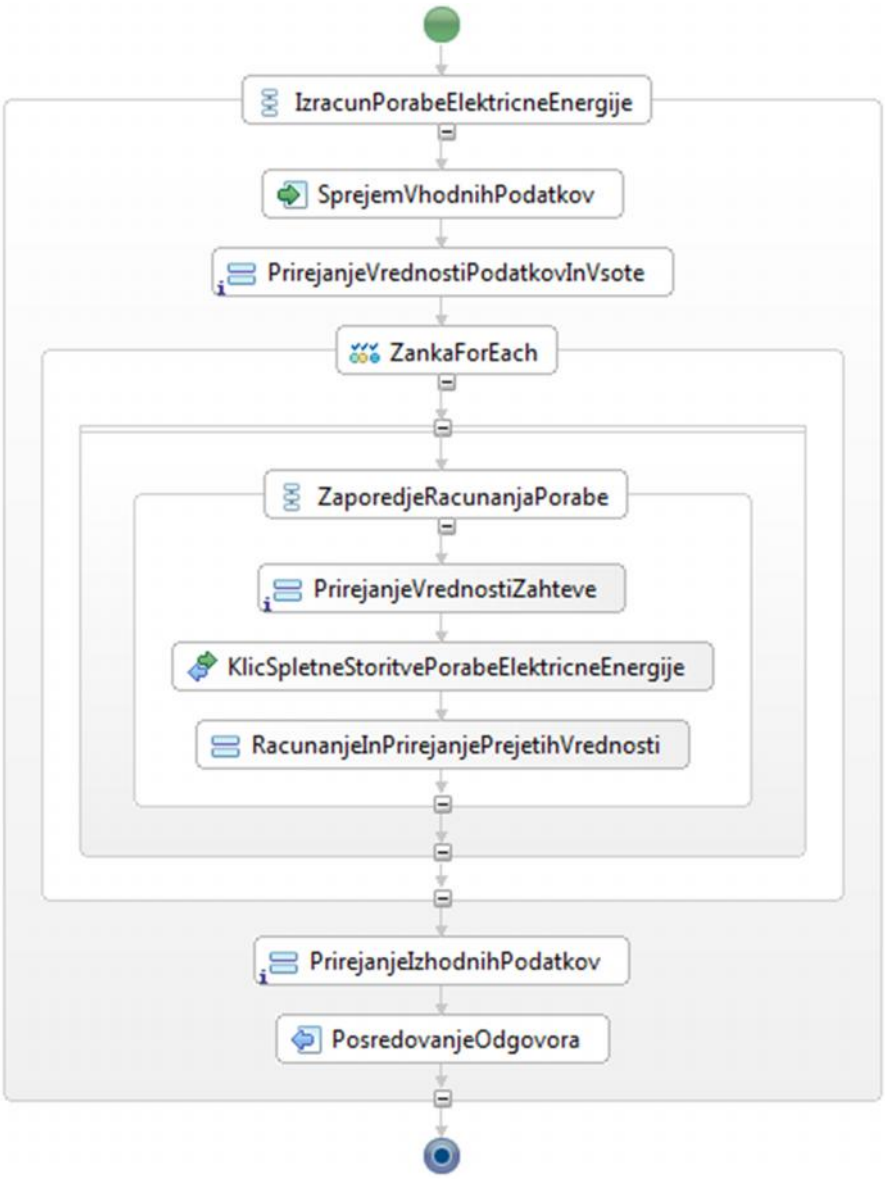
Slika 6.2: Poslovni proces preverjanja razpoložljivost izbrane telefonske številke.

Do razhajanj pri številu aktivnosti, merjenih z uporabo metrik, za razliko od predhodnega primera (spletne menjalnice) prihaja predvsem zaradi uporabe aktivnosti vejitve. Aktivnost vejitve je pri izvedbi poslovnega procesa preko spletne storitve SWF implementirana na drugačen način kot pri jeziku BPEL. Ključno razliko predstavlja aktivnost »Sequence«, ki poskrbi za zaporedno izvajanje gnezdenih aktivnosti in je prisotna pri aktivnosti vejitve v obeh primerih (če je pogoj izpolnjen ali ne – »if« in »else«). Pri implementaciji aktivnosti preko spletne storitve SWF aktivnost »Sequence« spustimo (poglavje 4.2.2.1) in jo nadomestimo z vsebino njenih gnezdenih aktivnosti, kar vpliva na rezultat metrik, prikazanih v tabeli 6.2. Omenjena preslikava povzroči zmanjšanje števila aktivnosti (NOA – 17 %), kar posledično privede do razlik tudi pri vseh ostalih metrikah (NOAC – 67 %). Funkcionalnost poslovnega procesa kljub omenjeni spremembi ostaja nespremenjena.

Metrika	Proces jezika BPEL v testnem okolju	Proces jezika BPEL preslikan na delovne tokove spletne storitve SWF	Razlika v %
NOA	12	10	17
NOAC	3	1	67
MCC	3	3	0
CFC	1	1	0

Tabela 6.2: Primerjava metrik procesa jezika BPEL in delovnih tokov spletne storitve SWF na primeru preverjanja razpoložljivost izbrane telefonske številke.

Podobne rezultate dosežemo tudi pri poslovnem procesu, ki podpira funkcionalnost izračuna stroška porabe električne energije na mesečnem nivoju za obdobje enega leta. Izračun porabe električne energije na letnem nivoju poteka v obliki zanke, kjer se za vsak mesec pridobijo podatki o porabi električne energije. Podatki so pridobljeni na podlagi zunanje spletne storitve za željen mesec. Rezultat poslovnega procesa predstavlja vsota porabe vseh mesecev, kar predstavlja letno porabo električne energije. Primer poslovnega procesa prikazuje slika 6.3.



Slika 6.3: Poslovni proces izračuna letne porabe električne energije.

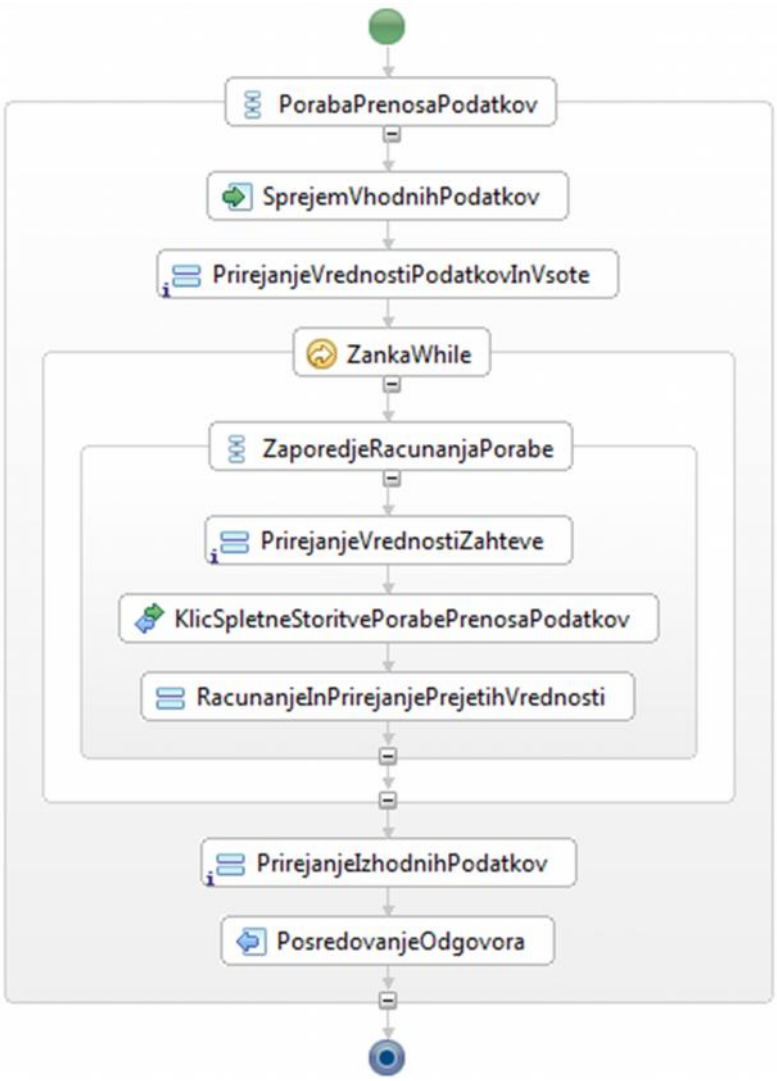
Pri izvedbi poslovnega procesa je uporabljena aktivnost, ki podpira pogojno ponavljajoče se segmente – »ForEach«. Podobno kot aktivnost vejitve tudi ta aktivnost za gnezdene aktivnosti uporablja funkcionalnost konstrukta »Sequence«, ki poskrbi za pravilno zaporedje ponavljajočih se segmentov. Poleg zaporedja pa aktivnost »ForEach« uporablja tudi gnezden konstrukt »Scope«, ki predstavlja funkcionalnost omejitve znotraj določenega obsega. Če je znotraj aktivnosti »Scope« uporabljenih več konstruktov, je treba za izvajanje le-teh določiti vrstni red, zato v tem primeru pride do gnezdenja aktivnosti »Sequence«. Podobno kot aktivnost »Sequence« je tudi aktivnost »Scope« pri izvedbi poslovnega procesa v okviru spletne storitve SWF implementirana na drugačen način v primerjavi z jezikom BPEL. V obeh primerih namesto omenjenih aktivnosti uporabimo njune gnezdene aktivnosti, kar ne vpliva na funkcionalnost poslovnega procesa. Sprememba implementacije posledično privede do spremembe števila aktivnosti in povezav, kar se odraža tudi pri različnih metrikah (tabela

6.3). Tako se število aktivnosti zmanjša za 19 %, medtem ko se število aktivnosti in kontrolnih poti zmanjša za 50 % oz. za 40 %. Funkcionalnost poslovnega procesa kljub spremembi števila aktivnosti ostaja nespremenjena.

Metrika	**Proces jezika BPEL v testnem okolju**	**Proces jezika BPEL preslikan na delovne tokove spletne storitve SWF**	**Razlika v %**
NOA	11	9	19
NOAC	2	1	50
MCC	5	3	40
CFC	0	0	0

Tabela 6.3: Primerjava metrik procesa jezika BPEL in delovnih tokov spletne storitve SWF na primeru izračuna porabe električne energije za obdobje enega leta.

Zadnji izmed testnih poslovnih procesov podpira funkcionalnost preverjanja porabe storitve prenosa podatkov mobilne naprave pri enem od ponudnikov mobilne telefonije. Primer poslovnega procesa prikazuje slika 6.4, podrobnejši opis funkcionalnosti pa je predstavljen v poglavju 5.4.



Slika 6.4: Poslovni proces izračuna porabe storitve prenosa podatkov mobilne naprave.

Podobno kot predhodni primer tudi ta vsebuje pogojno ponavljajoče se segmente. V tem primeru je funkcionalnost ponavljajočih se segmentov podprta z uporabo aktivnosti »While« jezika BPEL. Pri slednji velja podobno pravilo, ki pri izvedbi več kot ene gnezdene aktivnosti predvideva uporabo aktivnosti »Sequence«. Kot smo navedli v predhodnih primerih, je aktivnost »Sequence« pri izvedbi preko spletne storitve SWF implementirana kot zaporedje gnezdenih aktivnosti. V tem primeru se ta sprememba odraža v zmanjšanju števila aktivnosti za 12 % in števila kontrolnih tokov za 50 %, pa tudi števila povezav za 50 %. Posledice zmanjšanja aktivnosti posledično privede do različnih ovrednotenj pri uporabi metrik, kot je prikazano v tabeli 6.4. Podobno kot za ostale poslovne procese velja tudi za ta primer, da je ne glede na zmanjšanje števila aktivnosti sama funkcionalnost poslovnega procesa nespremenjena.

Metrika	Proces jezika BPEL v testnem okolju	Proces jezika BPEL preslikan na delovne tokove spletne storitve SWF	Razlika v %
NOA	9	8	12
NOAC	2	1	50
MCC	4	2	50
CFC	0	0	0

Tabela 6.4: Primerjava metrik procesa jezika BPEL in delovnih tokov spletne storitve SWF na primeru porabe storitve prenosa podatkov mobilne naprave.

6.3 Primerjava metrik testnih procesov

Pri vsakem izmed poslovnih procesov lahko vidimo, da se ovrednoteno število na podlagi metrike bistveno ne razlikuje med poslovnim procesom jezika BPEL, izvedenim na razvojnem okolju, in v obliki delovnih tokov spletne storitve SWF. Do razhajanj pri številu aktivnosti pride v primerih uporabe aktivnosti, ki podpirajo vejitve in pogojno ponavljajoče se segmente (aktivnosti «If«, »ForEach« in »While«). Razlike so vidne predvsem pri načinu implementacije posamezne aktivnosti, v naših primerih pri aktivnostih »Sequence« in »Scope«. Funkcionalnost poslovnih procesov zaradi omenjenih sprememb ni spremenjena. Potrditev enake funkcionalnosti smo opravili v sklopu verifikacije (v več fazah, kot je opisano v poglavju 6.1) na vseh štirih testnih primerih poslovnih procesov.

Če med seboj primerjamo vse testne poslovne procese, pridemo do spoznanja, da večjih odstopanj ni. Pomemben podatek predstavlja tudi dejstvo, da vsak izmed testnih poslovnih procesov podpira enako funkcionalnost na obeh okoljih kljub razlikam v sami implementaciji, kar je razvidno iz tabele 6.5. V primeru uporabe zgolj primitivnih aktivnosti razlike dejansko ni, saj se vsaka aktivnost jezika BPEL enolično preslika v svoj konstrukt na spletni storitvi SWF. Posledično v teh primerih odstopanj pri uporabi različnih metrik ni. Razlike so najbolj vidne v kompleksnejših primerih, kjer uporabljamo pogojno ponavljajoče se segmente, saj pride do razlike pri številu aktivnosti kot posledica razlike v implementaciji. V našem primeru najbolj odstopa testni primer izračuna porabe električne energije, kjer je zmanjšanje števila aktivnosti pri izvajanju v obliki delovnih tokov spletne storitve SWF najbolj vidno – 19 %. Podobno velja za primer razpoložljivosti telefonske številke, kjer pride do največjega odstopanja pri izračunu aktivnosti v kontrolnih tokovih. Razlika je več kot očitna zaradi uporabe pogojne vejitve, ki privede do zmanjšanja števila aktivnosti za kar 67 %. V primeru uporabe funkcionalno podobnih aktivnosti je tudi odstopanje pri uporabi metrik med tesnimi procesi minimalno oz. ga sploh ni. Tak primer vidimo pri primerjanju aktivnosti v kontrolnih tokovih na primerih izračuna porabe električne energije in izračuna prenosa podatkov mobilne naprave. V obeh primerih uporabljamo aktivnosti, ki vsebujejo pogojno ponavljajoče se

segmente, a se ti med seboj razlikujejo. Kljub različnim tipom aktivnosti je njuna funkcionalnost podobna. Enako velja pri implementaciji obeh konstruktov pri uporabi delovnih tokov spletne storitve SWF. Rezultat metrike je tako v obeh primerih enak, saj dosežemo enako zmanjšanje števila aktivnosti za kar 50 %. Seveda je število aktivnosti odvisno predvsem od funkcionalnosti poslovnega procesa, a v primeru testnih poslovnih procesov pri vseh metrikah v povprečju dosegamo manjše število aktivnosti pri izvajanju v obliki delovnih tokov spletne storitve SWF.

Metrika	Spletna menjalnica	Razpoložljivost telefonske številke	Izračun električne energije	Izračun prenosa podatkov	Povprečna razlika	Ekvivalentna funkcionalnost
NOA	0 %	17 %	19 %	12 %	12 %	DA
NOAC	0 %	67 %	50 %	50 %	41,7 %	DA
MCC	0 %	0 %	40 %	50 %	22,5 %	DA
CFC	0 %	0 %	0 %	0 %	0 %	DA

Tabela 6.5: Primerjava kompleksnosti pri testnem naboru poslovnih procesov.

Na podlagi izvedbe verifikacije in primerjave izvedbe poslovnih procesov v obliki metrik smo potrdili našo predpostavko o ohranjanju kompleksnosti poslovnih procesov. Naše izhodišče je predstavljal obstoječ proces jezika BPEL, na podlagi katerega smo izvedli preslikavo posameznih aktivnosti. Pri postopku preslikave posameznih aktivnosti smo določene konstrukte (»Sequence, »Scope«) izključili iz preslikave (poglavje 4.2.2.1). S tem smo dosegli zmanjšanje števila aktivnosti posameznega procesa, kar je razvidno tudi iz testnih poslovnih procesov. Pri preslikavi slednjih smo tako v povprečju pridelali za 12 % manj aktivnosti. S tem smo kompleksnost posameznega poslovnega procesa uspešno zmanjšali. Zmanjšanje kompleksnosti predstavlja dobro izhodišče za možno optimizacijo poslovnih procesov.

Doseganje nižjega števila aktivnosti pozitivno vpliva tudi na izvedbo poslovnega procesa v obliki delovnih tokov spletne storitve SWF. Implementacija vsake aktivnosti je izvedena na gostiteljskem računalniku, medtem ko je sama izvedba slednje opravljena na spletni storitvi SWF. Tak način izvajanja je sicer visoko razširljiv in močno zaželen. Kljub temu ima tak sistem tudi svoje slabosti. Namreč pri uporabi delovnih tokov spletne storitve SWF je določen delež stroškov uporabe med drugim odvisen tudi od števila izvedenih aktivnosti [7]. V našem primeru je pri izvajanju testnih poslovnih procesov ta aspekt zanemarljiv, vendar pri večji količini kompleksnejših poslovnih procesov to ni tako. Če predlagani koncept preslikave uporabljamo na dnevnem nivoju (tako, kot je to pri uporabi jezika BPEL), na naboru realnih poslovnih procesov lahko omenjen (stroškovni) vidik predstavlja resen problem. Vsekakor je zmanjšanje števila aktivnosti dobrodošlo tudi s strani zmanjšanja časa, potrebnega za izvedbo posameznega poslovnega procesa. Zmanjšanje izvajalnega časa bi posebej prišlo do izraza pri kompleksnejših procesih, kjer imamo večje število aktivnosti in gnezdenih aktivnosti na

različnih nivojih. Do časovne razlike pri izvajanju poslovnega procesa pride predvsem zaradi zmanjšanja števila aktivnosti, kar je posledica koncepta preslikave na delovni tok. Pri predlaganem konceptu se to najbolj odraža pri preslikavi različnih gnezdenih aktivnostih, ki so pri kompleksnejših aktivnostih kar pogoste.

7 Zaključek

Koncept izvajanja poslovnih procesov predstavlja tehnološko zapleten postopek. Posledično je izvajanje poslovnih procesov pogojeno s porabo velikega nabora virov s področja informacijskih tehnologij. S prihodom koncepta računalništva v oblaku se pogled na porabo glede na razpoložljivost računalniških virov spremeni. Računalniški oblak omogoča izrabo različnih računalniških virov glede na potrebe posameznika. Tak koncept predstavlja dobro izhodišče za izvedbo zapletenih postopkov, kot so na primer poslovni procesi. Trenutno nihče od obstoječih ponudnikov oblačnih storitev ne podpira možnosti izvajanja poslovnih procesov v okviru računalniškega oblaka. V ta namen smo v sklopu izdelave magistrske naloge predlagali razširitev okolja delovanja poslovnih procesov in njihovo premestitev z lokalnih okvirov na okvire računalniškega oblaka. Ta razširitev je vključevala izdelavo modela preslikave obstoječih konstruktov poslovnega procesa jezika BPEL na delovne tokove spletne storitve SWF računalniškega oblaka podjetja Amazon. Rezultat magistrske naloge je poleg izdelave tudi vrednotenje prototipne aplikacije, ki podpira predlagano razširitev in uspešno izvedbo poslovnih procesov na platformi računalniškega oblaka.

Na področju preslikave poslovnih procesov je bilo v zadnjih letih vloženega veliko dela. Posledično so nastali številni koncepti in preslikave na različne modele. Najbolj znana je preslikava med jezikom BPEL in notacijo BPMN, ki predstavlja premostitev med konceptualnim neskladjem jezika ter notacije, kar poveča samo kompleksnost preslikave. Določene aktivnosti jezika BPEL ne moremo neposredno preslikati na notacijo BPMN, saj podobni konstrukti pri omenjeni notaciji ne obstajajo. Drug pristop k preslikavi jezika BPEL predvideva uporabo modela Petrijevih mrež. Te predstavljajo enega od mnogih matematičnih modelirnih jezikov, ki služijo opisu porazdeljenih sistemov. Preslikava jezika BPEL na model Petrijevih mrež omogoča grafično notacijo stopenjskih procesov in obenem podpira vse potrebne funkcionalnosti (zaporedje izvajanja, vejitve, pogojno ponavljajoče se segmente, paralelno izvajanje). Na tem področju je bilo izdelanih veliko preslikav, a najbolj znani sta dve, in sicer preslikava v OWFN in WFN. Pri obeh tipih preslikav naletimo na določene pomanjkljivosti, ki predstavljajo omejitve in niso najbolj primerne za apliciranje.

Nobena od rešitev ni omogočala primerne preslikave v celoti, zato smo se odločili za izdelavo lastne preslikave, ki se bo kar najbolj prilegala konceptu delovnih tokov spletne storitve SWF. V ta namen smo predvideli preslikavo vseh relevantnih aktivnosti jezika BPEL. Preslikavo smo podprli z izdelavo prototipne aplikacije, ki je predstavljala potrditev koncepta. Kot končni rezultat preslikave smo si zadali izvedbo poslovnega procesa jezika BPEL v obliki delovnega toka spletne storitve SWF. Tako smo se pri izvedbi preslikave osredotočili predvsem na osnovne konstrukte jezika BPEL, ki so največkrat v uporabi in jih tudi uspešno preslikali. Vsako izmed relevantnih aktivnosti jezika BPEL smo posebej analizirali, kar je

služilo kot izhodišče za preslikavo. Na podlagi rezultatov analize smo vsako aktivnost posebej preslikali v okvir spletne storitve SWF. Neposredno preslikavo smo izvedli v primeru enostavnih aktivnosti, ki niso vsebovale naprednih konceptov. Pri preslikavi kompleksnih aktivnosti smo uporabili drugačne pristope in njihovo implementacijo preslikali kot skupek enostavnih aktivnosti, v obliki rekurzivnih klicev ali podtokov. Rezultat preslikave posamezne aktivnosti je predstavljal konstrukt delovnega toka spletne storitve SWF, ki je funkcionalno podprl enako aktivnost jezika BPEL. Preslikave poslovnih procesov smo podprli v obliki prototipne aplikacije. Slednja je kot vhodni podatek pridobila poslovni proces jezika BPEL in ga s pomočjo preslikave posameznih vsebovanih aktivnosti uspešno izvedla na računalniškem oblaku v obliki delovnega toka spletne storitve SWF. Rezultat in postopek izvedbe poslovnega procesa je bil viden v okviru nadzorne plošče omenjene spletne storitve.

Z uspešno izvedbo poslovnih procesov v obliki delovnega toka spletne storitve SWF smo potrdili zastavljen koncept. S tem smo potrdili tudi našo hipotezo o možnosti izvedbe poslovnih procesov jezika BPEL na računalniškem oblaku. Tako smo na podlagi izdelane prototipne aplikacije uspešno izvedli določen nabor testnih poslovnih procesov. Uspešnost izvedbe posameznih procesov smo merili na različne načine, kjer smo primerjali: način izvedbe (na lastnem razvojnem okolju in na spletni storitvi SWF), izvedbo enakih aktivnosti, pa tudi končni rezultat. Poleg izvajanja poslovnih procesov smo primerjali tudi preslikavo aktivnosti na obeh okoljih. Pri testnem naboru poslovnih procesov smo prišli do zanimivih spoznanj, saj so bili rezultati vseh primerov, izvedenih na obeh okoljih, enaki. Posamezni poslovni procesi so bili v obliki delovnih tokov spletne storitve SWF predstavljeni z manjšim številom konstruktov (v povprečju 12 % manj aktivnosti) v primerjavi z jezikom BPEL, kar je posledica implementacije preslikave in predstavlja dobro osnovo v primeru optimizacije poslovnih procesov. Kljub zmanjšanju števila aktivnosti pri uporabi delovnih tokov spletne storitve SWF je funkcionalnost poslovnega procesa ostala nespremenjena, kar je posledično pripeljalo do enakih končnih rezultatov.

Pri izdelavi magistrske naloge smo se osredotočili na osnovne gradnike in tiste, ki so največkrat v uporabi. Napredni koncepti, kot so razni upravljavci dogodkov, prekinitev napak in kompenzacij, v tem delu niso obravnavani, zato ostaja precej izzivov in prostora za izboljšave. Pri uporabi delovnih tokov spletne storitve SWF smo naleteli tudi na določene omejitve, ki omejujejo naš koncept in bi ga bilo treba pravilno razširiti, če bi želeli izkoriščati poln nabor funkcionalnosti. Primer ene takih je tudi omejitev velikosti podatkov, ki se prenašajo med odjemalcem in spletno storitvijo. Premostitev omenjene težave je izvedljiva z uporabo podatkovne baze v okviru računalniškega oblaka (npr. Dynamo DB). V tem primeru so vhodni podatki delovnih tokov spletne storitve SWF posredovani neposredno s strani podatkovne baze (namesto s strani odjemalca), obenem pa bi s to premostitvijo dosegli tudi boljši odzivni čas izvajanja poslovnih procesov.

Veliko prednost predlaganega koncepta preslikave vidimo v možnosti selitve obstoječih poslovnih procesov z okvirov lokalnega okolja na računalniški oblak. Pri tem lahko izvedemo migracijo obstoječih poslovnih procesov v obliki preslikav in s tem njihovo izvajanje preselimo na računalniški oblak. S selitvijo na računalniški oblak lahko na relativno enostaven način podpremo tudi druge aspekte poslovnih procesov z uporabo storitev, ki so na voljo v okviru računalniškega oblaka. S tem lahko že tako široko paleto ponudbe storitev računalniškega oblaka dodatno obogatimo z možnostjo izvajanja poslovnih procesov v okviru oblaka in jo končnim uporabnikom ponudimo tudi v obliki spletne storitve.

Literatura

[1] W.M.P. van der Aalst, The application of Petri nets to workflow management, Journal of Circuits, Systems and Computers št. 8, zv. 1, str. 21–66, 1998.

[2] W. M. P. van der Aalst, K. M. van Hee, A. H. M. ter Hofstede, N. Sidorova, H. M. W. Verbeek, M. Voorhoeve, M. T. Wynn, Soundness of Workow Nets: Classifcation, Decidability, and Analysis, Formal Aspects of Computing, št. 23, zv. 3 , str 333-363, 2011.

[3] W. M. P. van der Aalst, K. B.Lassen, Translating unstructured workflow processes to readable BPEL: Theory and implementation, Information and Software Technology, št. 50, zv. 3, str 131–159, 2008.

[4] E. Alemneh, A. A. A. Ghani, R. Atan, SAT4BSC: A Static Analysis Tool for BPEL Source Codes,International Journal of Computer Science and Mobile Computing, št.3 zv.2, str. 659–665, februar 2014.

[5] Activity implementation - AWS Flow Framework for Java, dostopno na: http://docs.aws.amazon.com/amazonswf/latest/awsflowguide/activityimpl.html (pridobljeno 20. 8. 2014).

[6] About AWS, dostopno na: https://aws.amazon.com/about-aws/, (pridobljeno 20. 3. 2015).

[7] AWS| Amazon Simple Workflow (SWF) Pricing, dostopno na: http://aws.amazon.com/swf/pricing/, pridobljeno (20. 3. 2015).

[8] J. Billington, M. Diaz, G. Rozenberg (Eds.), Application of Petri Nets to Communication Networks, Springer-Verlag Berlin Heidelberg 1999.

[9] Bussines Process Modeling Notation Specification, dostopno na: http://www.omg.org/bpmn/Documents/OMG_Final_Adopted_BPMN_1-0_Spec_06-02-01.pdf (13. 3. 2015).

[10] Bussines Process Management, Inc - What is BPM?, dostopno na: http://bpm.com/what-is-bpm, (pridobljeno 20. 3. 2015).

[11] Customizing JAXB bindings, Dostopno na: http://docs.oracle.com/cd/E17802_01/webservices/webservices/docs/1.5/tutorial/doc/JAXBUsing4.html (pridobljeno 5. 10. 2014).

[12] G. Decker, J. Mendling, Instantiation Semantics for Process Models. Business Process Management: 6th International Conference, BPM 2008, Milan, Italy, September 2-4, 2008. Proceedings, zv. 5240, str. 164–179, Springer Berlin Heidelberg, 2008.

[13] Dijkman, R., Dumas, M., Ouyang, C.: Semantics and Analysis of Business Process Models in BPMN, Information and Software Technology, št. 50, zv. 12, str. 1281–1294, November 2008.

[14] A. Frece, M. B. Juric, Modeling functional requirements for configurable content- and context-aware dynamic service selection in business process models, Journal of Visual Languages & Computing, št. 23, zv. 4, str. 223–247, avgust 2012.

[15] R. Heckel, Open Petri Nets as Semantic Model for Workflow Integration, str 281–194, Springer Berlin Heidelberg, 2003.

[16] K. Jensen, Coloured Petri Nets Basic Concepts, Analysis Methods and Practical Use. Volume 1, str. 65–87, Springer-Verlag Berlin Heidelberg, 1996.

[17] M. Jurišić, Transition between process models (BPMN) and service models (WS-BPEL and other standards): A systematic review, Journal of Information and Organizational Sciences, št. 35, zv. 2, str. 163–171, december 2011.

[18] JAXB Release Documentation, dostopno na: https://jaxb.java.net/2.2.11/docs/ (pridobljeno 5. 10. 2014).

[19] JAX-WS Release Documentation, dostopno na: https://jax-ws.java.net/2.2.10/docs/ (pridobljeno 7. 9. 2014).
[20] N. Lohmann, P. Massuthe, C. Stahl, D. Weinberg, Analyzing interacting WS-BPEL processes using flexible model generation, Data & Knowledge Engineering, št. 64, zv. 1, str 38–54, 2008.
[21] N. Lohmann, A Feature-Complete Petri Net Semantics for WS-BPEL 2.0. Web Services and Formal Methods: 4th International Workshop, WS-FM 2007, Brisbane, Australia, September 28-29, 2007. Proceedings, zv. 4937, str. 77–91, Springer Berlin Heidelberg, 2008.
[22] Maven Introduction, dostopno na: http://maven.apache.org/what-is-maven.html, (pridobljeno 14. 9. 2014).
[23] Maven - POM Reference, dostopno na: http://maven.apache.org/pom.html#What_is_the_POM, (pridobljeno 14. 9. 2014).
[24] P. Mell, T.Grance, The NIST Definition of Cloud Computing, National Institute of Standards & Technology Gaithersburg, MD, United States, 2011.
[25] T. Murata, Petri nets: Properties, analysis and applications, Proceedings of the IEEE, št. 77, zv. 4, str. 541–580, IEEE, 1989.
[26] Oasis, dostopno na: https://www.oasis-open.org/org, (pridobljeno 7. 2. 2015).
[27] OASIS Web Services Business Process Execution Language (WSBPEL) TC Dostopno na: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel, (pridobljeno 17. 2. 2015).
[28] C. Ouyang, H.M.W. Verbeek, W.M.P. van der Aalst, S. Breutel, M. Dumas, A.H.M. ter Hofstede, Formal semantics and analysis of control flow in WS-BPEL, Science of Computer Programming, št. 67, zv. 2–3, str. 162–198, 2007.
[29] C. Ouyang, W.M.P. van der Aalst,M. Dumas1, A.H.M. ter Hofstede1, From BPMN Process Models to BPEL Web Services. ICWS '06 Proceedings of the IEEE International Conference on Web Services (ICWS'06), str. 285–192, IEEE, 2006.
[30] Overwiew AWS SDK for Java 1.9.26, dostopno na: http://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/index.html, (pridobljeno 7. 9. 2014).
[31] C. Pautesso, Web service composition with BPEL for REST, Journal Data & Knowledge Engineering, št. 68, zv 9, str. 851–866, 2009.
[32] J.L. Peterson, Petri Net Theory and the Modeling of Systems, Prentice Hall, 1981.
[33] J. Recker, J. Mendling, On the translation between BPMN and BPEL: Conceptual mismatch between process modeling languages. In Eleventh Int. Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD'06), 521–532, Namur University Press, 2006.
[34] K. Salimifard, M. Wright, Petri net-based modelling of workflow system: An overview, European Journal of Operational Research, št. 143, zv. 3, str 664–676, 2001.
[35] D. Schumm, D. Karastoyanova, F. Leymann, J. Nitzsche, On Visualizing and Modelling BPEL with BPMN, Grid and Pervasive Computing Conference, 2009. GPC '09. Workshops at the, str. 80–87, IEEE, 2009.
[36] SOAP Specifications, dostopno na: http://www.w3.org/TR/soap/, (pridobljeno 10. 3. 2015).
[37] H. M. W. Verbeek, Analyzing BPEL processes using Petri nets, str. 59–78, Florida International University, 2005.
[38] J. Vanhatalo, H. Völzer, and J. Koehler. The Refined Process Structure Tree. Business Process Management: 6th International Conference, BPM 2008, Milan, Italy, September 2-4, 2008. Proceedings, str. 100–115, Springer Berlin Heidelberg, 2008.

[39] Web Service Bussiness Process Execution Language, dostopno na: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html#_Toc164738496, (pridobljeno 23. 9. 2014).
[40] Web Services Description Language for Java. Dostopno na: http://wsdl4j.sourceforge.net/, (pridobljeno 23. 8. 2014).
[41] M. Weidlich, G. Decker, A. Großkopf, M. Weske, BPEL to BPMN: The Myth of a Straight-Forward Mapping. On the Move to Meaningful Internet Systems: OTM 2008: OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008, Monterrey, Mexico, November 9-14, 2008, Proceedings, Part I, zv. 5331, str 265–282, Springer Berlin Heidelberg, 2008.
[42] What is the AWS Flow Framework for Java? Dostopno na: http://docs.aws.amazon.com/amazonswf/latest/awsflowguide/welcome.html, (pridobljeno 17. 2. 2015).
[43] A.Wombacher, A.Martens, Soundness and Equivalence of Petri Nets and annotated Finite State Automate:A Comparison in the SOA context, Digital EcoSystems and Technologies Conference, 2007. DEST '07. Inaugural IEEE-IES, str. 257–262, IEEE, 2007.
[44] W. Zhao, Y. Huang, C. Yuan, L. Wang, Formalizing Business Process Execution Language Based on Petri Nets, Intelligent Systems and Applications (ISA), 2010 2nd International Workshop on, str 1–8, IEEE, 2010.